

Reasoning with Kan fillings about Morse reductions

Formalisation of mathematics with interactive theorem provers
Cambridge University

10 October 2024

Maximilian Doré
`maximilian.dore@cs.ox.ac.uk`

Homotopy type theory in practice

Cubical type theory implements *homotopy type theory*, which views equalities in a type as *paths/homotopies* in a topological space.

Cubical type theory is a ...

- **logic for homotopy types:** types have higher structure, and properties of cubical sets turn into reasoning principles, e.g., *Kan filling*.
- **programming language:** higher inductive types (e.g., quotients) and univalence are computationally well-behaved.

Homotopy type theory in practice

Cubical type theory implements *homotopy type theory*, which views equalities in a type as *paths/homotopies* in a topological space.

Cubical type theory is a ...

- **logic for homotopy types:** types have higher structure, and properties of cubical sets turn into reasoning principles, e.g., *Kan filling*.
- **programming language:** higher inductive types (e.g., quotients) and univalence are computationally well-behaved.

In this talk:

- how to automate Kan filling.
- how to formalise *discrete Morse theory*, method from computational topology.

Homotopy type theory in practice

Cubical type theory implements *homotopy type theory*, which views equalities in a type as *paths/homotopies* in a topological space.

Cubical type theory is a ...

- **logic for homotopy types:** types have higher structure, and properties of cubical sets turn into reasoning principles, e.g., *Kan filling*.
- **programming language:** higher inductive types (e.g., quotients) and univalence are computationally well-behaved.

In this talk:

- how to automate Kan filling.
- how to formalise *discrete Morse theory*, method from computational topology.

Make cubical type theory more usable, and make use of it.

Automatically deriving Kan fillings

jww Evan Cavallo & Anders Mörtberg

Cubes and Kan fillings

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. Consider *cell contexts* capturing a single type.

Cubes and Kan fillings

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. Consider *cell contexts* capturing a single type.

$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

$$\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$$

Cubes and Kan fillings

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. Consider *cell contexts* capturing a single type.

$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

$$\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$$

$$x \xrightarrow{x} x \xrightarrow{i}$$

$$\Gamma \mid i \vdash x : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x]$$

Cubes and Kan fillings

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. Consider *cell contexts* capturing a single type.

$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

$$\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$$

$$x \xrightarrow{x} x \xrightarrow{i}$$

$$\Gamma \mid i \vdash x : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x]$$

“ x solves *boundary problem* $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x]$ ”

Cubes and Kan fillings

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. Consider *cell contexts* capturing a single type.

$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

$$\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$$

$$x \xrightarrow{x} x \xrightarrow{i}$$

$$\Gamma \mid i \vdash x : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x]$$

“ x solves *boundary problem* $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x]$ ”

$$y \overset{?}{\dashrightarrow} x$$

$$\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto x]$$

Cubes and Kan fillings

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. Consider *cell contexts* capturing a single type.

$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

$$\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$$

$$x \xrightarrow{x} x \xrightarrow{i}$$

$$\Gamma \mid i \vdash x : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x]$$

“ x solves boundary problem $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x]$ ”

$$\begin{array}{ccc} y & \overset{?}{\dashrightarrow} & x \\ q(j) \uparrow & & \uparrow x \\ & \xrightarrow{x} & \end{array} \quad \begin{array}{c} j \\ \uparrow \\ i \end{array}$$

$$\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto x]$$

Cubes and Kan fillings

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. Consider *cell contexts* capturing a single type.

$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

$$\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$$

$$x \xrightarrow{x} x \xrightarrow{i}$$

$$\Gamma \mid i \vdash x : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x]$$

“ x solves boundary problem $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x]$ ”

$$\begin{array}{ccc} y & \overset{?}{\dashrightarrow} & x \\ q(j) \uparrow & & \uparrow x \\ & \xrightarrow{x} & \\ & & i \end{array} \quad \begin{array}{c} j \\ \uparrow \\ i \end{array}$$

$$\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto x]$$

$$? \triangleq \text{fill} \left(\begin{array}{ccc} & \uparrow & \uparrow \\ q(j) & \lrcorner & x \\ & \xrightarrow{x} & \end{array} \right)$$

Constructing open cubes with constraint solving

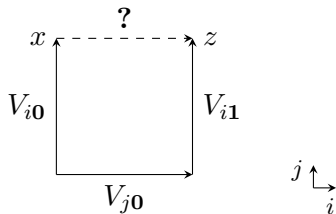
$\Gamma \triangleq x : [], y : [], z : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$

Boundary problem: $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto z]$

Constructing open cubes with constraint solving

$$\Gamma \triangleq x : [], y : [], z : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$$

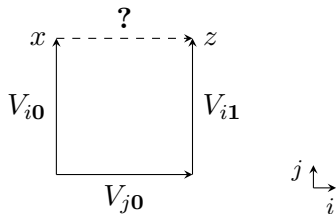
Boundary problem: $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto z]$



Constructing open cubes with constraint solving

$\Gamma \triangleq x : [], y : [], z : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$

Boundary problem: $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto z]$



Variables have domains:

- $D_{i0} = \{x\}$
- $D_{i1} = \{z, r(j)\}$
- $D_{j0} = \{x, y, z, q(i), r(i)\}$

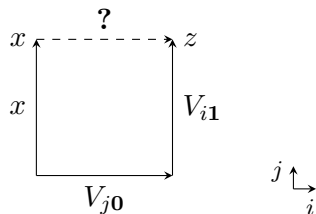
Subject to constraints:

- $V_{i0}[j = \mathbf{0}] = V_{j0}[i = \mathbf{0}]$
- $V_{i1}[j = \mathbf{0}] = V_{j0}[i = \mathbf{1}]$

Constructing open cubes with constraint solving

$\Gamma \triangleq x : [], y : [], z : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$

Boundary problem: $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto z]$



Variables have domains:

- $D_{i0} = \{x\}$
- $D_{i1} = \{z, r(j)\}$
- $D_{j0} = \{x, y, z, q(i), r(i)\}$

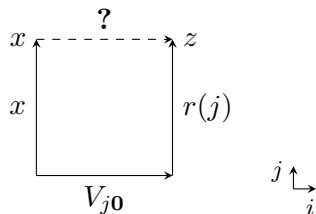
Subject to constraints:

- $V_{i0}[j = \mathbf{0}] = V_{j0}[i = \mathbf{0}]$
- $V_{i1}[j = \mathbf{0}] = V_{j0}[i = \mathbf{1}]$

Constructing open cubes with constraint solving

$\Gamma \triangleq x : [], y : [], z : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$

Boundary problem: $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto z]$



Variables have domains:

- $D_{i0} = \{x\}$
- $D_{i1} = \{z, r(j)\}$
- $D_{j0} = \{x, y, z, q(i), \cancel{r(i)}\}$

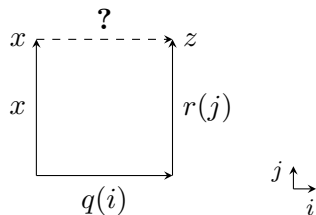
Subject to constraints:

- $V_{i0}[j = 0] = V_{j0}[i = 0]$
- $V_{i1}[j = 0] = V_{j0}[i = 1]$

Constructing open cubes with constraint solving

$\Gamma \triangleq x : [], y : [], z : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$

Boundary problem: $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto z]$



Variables have domains:

- $D_{i\mathbf{0}} = \{x\}$
- $D_{i\mathbf{1}} = \{z, r(j)\}$
- $D_{j\mathbf{0}} = \{x, y, z, q(i), r(i)\}$

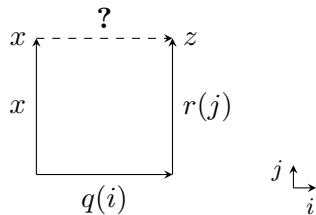
Subject to constraints:

- $V_{i\mathbf{0}}[j = \mathbf{0}] = V_{j\mathbf{0}}[i = \mathbf{0}]$
- $V_{i\mathbf{1}}[j = \mathbf{0}] = V_{j\mathbf{0}}[i = \mathbf{1}]$

Constructing open cubes with constraint solving

$\Gamma \triangleq x : [], y : [], z : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$

Boundary problem: $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto z]$



Variables have domains:

- $D_{i\mathbf{0}} = \{x\}$
- $D_{i\mathbf{1}} = \{z, r(j)\}$
- $D_{j\mathbf{0}} = \{x, y, z, q(i), \cancel{r(i)}\}$

Subject to constraints:

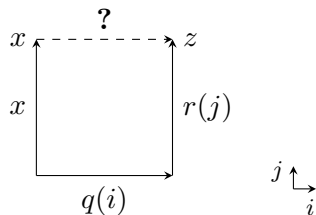
- $V_{i\mathbf{0}}[j = \mathbf{0}] = V_{j\mathbf{0}}[i = \mathbf{0}]$
- $V_{i\mathbf{1}}[j = \mathbf{0}] = V_{j\mathbf{0}}[i = \mathbf{1}]$

$$? \triangleq \text{fill}\left(\begin{array}{c} \uparrow x \\ \text{---} \\ \uparrow r(j) \\ \text{---} \\ \uparrow q(i) \end{array} \right)$$

Constructing open cubes with constraint solving

$\Gamma \triangleq x : [], y : [], z : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$

Boundary problem: $\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto z]$



Variables have domains:

- $D_{i\mathbf{0}} = \{x\}$
- $D_{i\mathbf{1}} = \{z, r(j)\}$
- $D_{j\mathbf{0}} = \{x, y, z, q(i), r(\cancel{i})\}$

Subject to constraints:

- $V_{i\mathbf{0}}[j = \mathbf{0}] = V_{j\mathbf{0}}[i = \mathbf{0}]$
- $V_{i\mathbf{1}}[j = \mathbf{0}] = V_{j\mathbf{0}}[i = \mathbf{1}]$

$$? \triangleq \text{fill} \left(\begin{array}{c} x \uparrow \\ \downarrow q(i) \end{array} \begin{array}{c} \uparrow r(j) \end{array} \right)$$

Find Kan fillings with finite domain constraint satisfaction programming.

Constructing nested Kan fillings

$$\Gamma \triangleq p(i) : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto x], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$$

$$\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto z]$$

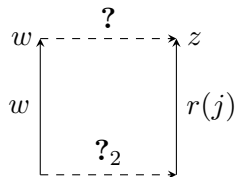
$$w \overset{?}{\dashrightarrow} z$$

$$\begin{array}{c} j \uparrow \\ \downarrow i \end{array}$$

Constructing nested Kan fillings

$$\Gamma \triangleq p(i) : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto x], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$$

$$\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto z]$$



$$? \triangleq \text{fill} \left(w \begin{array}{c} \uparrow \\ \text{?}_2 \\ \downarrow \end{array} r(j) \right)$$

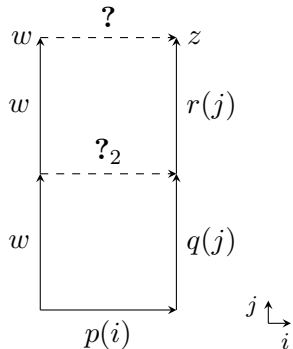
$$\text{where } \Gamma \mid i \vdash ?_2 : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto y]$$

$$j \uparrow \downarrow i$$

Constructing nested Kan fillings

$$\Gamma \triangleq p(i) : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto x], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$$

$$\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto z]$$



$$? \triangleq \text{fill} \left(w \begin{array}{c} \uparrow \\ \text{?}_2 \\ \downarrow \end{array} r(j) \right)$$

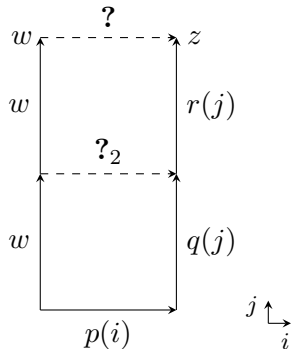
$$\text{where } \Gamma \mid i \vdash ?_2 : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto y]$$

$$?_2 \triangleq \text{fill} \left(w \begin{array}{c} \uparrow \\ \text{?}_2 \\ \downarrow \end{array} q(j) \right)$$

Constructing nested Kan fillings

$$\Gamma \triangleq p(i) : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto x], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y], r(i) : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto z]$$

$$\Gamma \mid i \vdash ? : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto z]$$



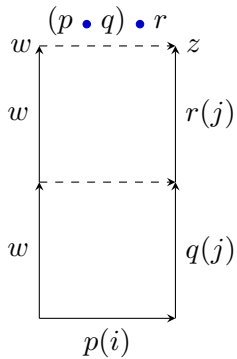
$$? \triangleq \text{fill} \left(w \begin{array}{c} \uparrow \\ \text{?}_2 \\ \downarrow \end{array} r(j) \right)$$

$$\text{where } \Gamma \mid i \vdash ?_2 : [i = \mathbf{0} \mapsto w \mid i = \mathbf{1} \mapsto y]$$

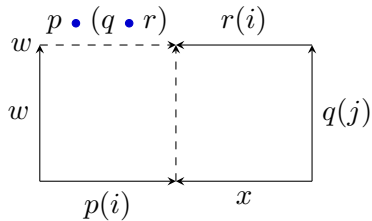
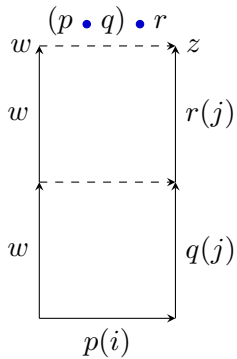
$$?_2 \triangleq \text{fill} \left(w \begin{array}{c} \uparrow \\ \text{?}_2 \\ \downarrow \end{array} q(j) \right)$$

Leave sides open if necessary, solve these as separate boundary problems.

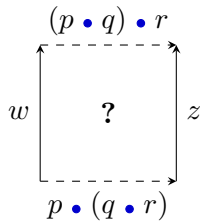
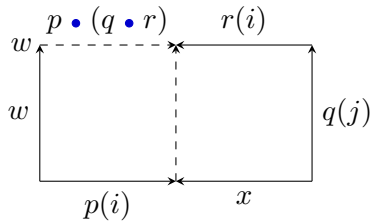
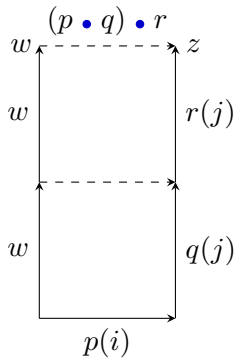
Associativity of path composition



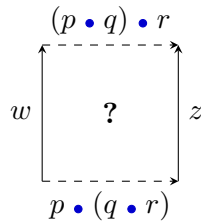
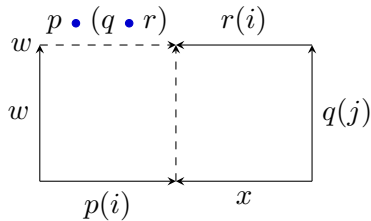
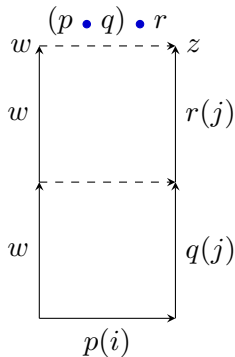
Associativity of path composition



Associativity of path composition



Associativity of path composition



→ construct an open 3-dimensional cube with ? on missing side.

Solving boundary problems automatically

Implementation: <https://github.com/maxdore/dedekind>
(WIP: proper integration in [Cubical Agda](#))

- Solver also takes into account certain admissible rules called *contortions*.
- Associativity of path composition solved in a few ms.
- Eckmann-Hilton solved in less than a second.

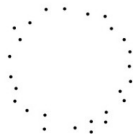
Kan filling summed up

- Solver which constructs Kan fillings based on constraint satisfaction programming and “breadth-first” search.
- Works for any cubical type theory currently being considered.
- Boundary problem in general undecidable and very difficult in higher dimensions, but in lower dimensions solver quickly carries out fiddly constructions.

Discrete Morse theory in Cubical Agda

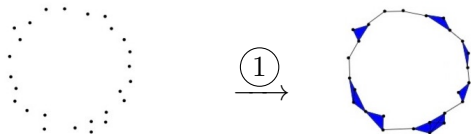
Topological data analysis

Study the *shape of data* using algebraic topology. Typical pipeline:



Topological data analysis

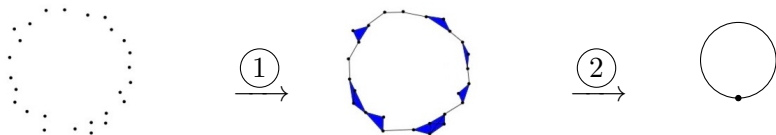
Study the *shape of data* using algebraic topology. Typical pipeline:



① associate *Vietoris-Rips* complex with dataset

Topological data analysis

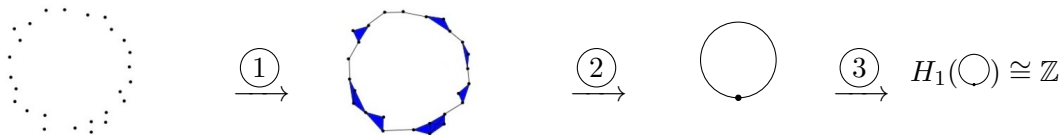
Study the *shape of data* using algebraic topology. Typical pipeline:



- ① associate *Vietoris-Rips* complex with dataset
- ② reduce size of complex using *discrete Morse theory*

Topological data analysis

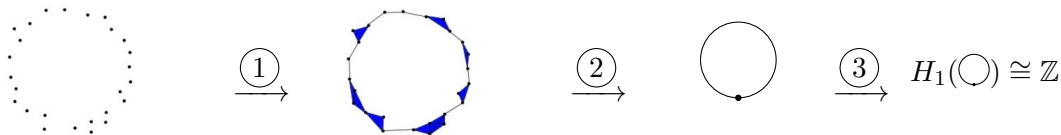
Study the *shape of data* using algebraic topology. Typical pipeline:



- ① associate *Vietoris-Rips* complex with dataset
- ② reduce size of complex using *discrete Morse theory*
- ③ compute (*persistent*) *homology*

Topological data analysis

Study the *shape of data* using algebraic topology. Typical pipeline:

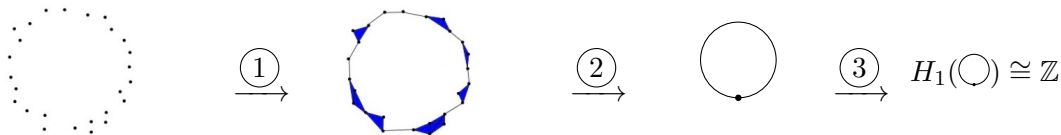


- ① associate *Vietoris-Rips* complex with dataset
- ② reduce size of complex using *discrete Morse theory*
- ③ compute (*persistent*) *homology*

Long-term goal: implement and verify this pipeline in [Cubical Agda](#).

Topological data analysis

Study the *shape of data* using algebraic topology. Typical pipeline:



- ① associate *Vietoris-Rips* complex with dataset
- ② reduce size of complex using *discrete Morse theory*
- ③ compute (*persistent*) *homology*

Long-term goal: implement and verify this pipeline in [Cubical Agda](#).

Today: ② for simple spaces, namely graphs

Showing Morse reductions correct

Discrete Morse theory removes cells irrelevant for the topology of a complex K , using an *acyclic partial matching* μ that says how to collapse cells.

Morse complex M contains only cells not part of μ .




Morse theorem: K and M are *equivalent as spaces*.

Showing Morse reductions correct

Discrete Morse theory removes cells irrelevant for the topology of a complex K , using an *acyclic partial matching* μ that says how to collapse cells.

Morse complex M contains only cells not part of μ .

Morse theorem: K and M are *equivalent as spaces*.




E.g., given  and apt matching, Morse theorem says  and  are equivalent.

Showing Morse reductions correct

Discrete Morse theory removes cells irrelevant for the topology of a complex K , using an *acyclic partial matching* μ that says how to collapse cells.

Morse complex M contains only cells not part of μ .

Morse theorem: K and M are *equivalent as spaces*.

E.g., given  and apt matching, Morse theorem says  and  are equivalent.

Original work¹ up to homology, modern approach² up to homotopy equivalence.

¹ Forman 1998, *Morse Theory for Cell Complexes*




² Nanda 2019, *Discrete Morse Theory and Localization*

Showing Morse reductions correct

Discrete Morse theory removes cells irrelevant for the topology of a complex K , using an *acyclic partial matching* μ that says how to collapse cells.

Morse complex M contains only cells not part of μ .

Morse theorem: K and M are *equivalent as spaces*.

E.g., given  and apt matching, Morse theorem says  and  are equivalent.

Original work¹ up to homology, modern approach² up to homotopy equivalence.

Formalise in [Cubical Agda](#) for arbitrary graph G :

- Characterise acyclic partial matchings μ and Morse complex M .
- Take geometric realisation $|G|, |M| : \text{Type}$ using higher inductive type.
- Prove that $|M| \equiv |G|$.

¹ Forman 1998, *Morse Theory for Cell Complexes*

² Nanda 2019, *Discrete Morse Theory and Localization*

Discrete Morse theory for graphs

A **partial matching** μ matches edges to an incident vertex.

μ is **acyclic** if all sequences of matched pairs end at an unmatched vertex.

Discrete Morse theory for graphs

A **partial matching** μ matches edges to an incident vertex.

μ is **acyclic** if all sequences of matched pairs end at an unmatched vertex.

$$M_0 \triangleq \Sigma u \notin \mu, \quad M_1(c, d) \triangleq \Sigma(uv \notin \mu).(u \rightsquigarrow c) \times (v \rightsquigarrow d)$$

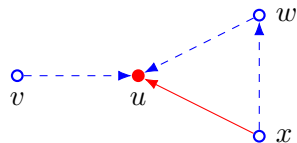
Discrete Morse theory for graphs

A **partial matching** μ matches edges to an incident vertex.

μ is **acyclic** if all sequences of matched pairs end at an unmatched vertex.

$$M_0 \triangleq \Sigma u \notin \mu, \quad M_1(c, d) \triangleq \Sigma(uv \notin \mu).(u \rightsquigarrow c) \times (v \rightsquigarrow d)$$

Example:



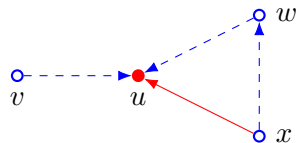
Discrete Morse theory for graphs

A **partial matching** μ matches edges to an incident vertex.

μ is **acyclic** if all sequences of matched pairs end at an unmatched vertex.

$$M_0 \triangleq \Sigma u \notin \mu, \quad M_1(c, d) \triangleq \Sigma(uv \notin \mu).(u \rightsquigarrow c) \times (v \rightsquigarrow d)$$

Example:



$$\mu \triangleq (v, vu), (w, wu), (x, xw)$$

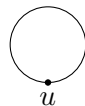
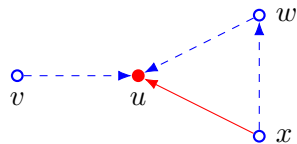
Discrete Morse theory for graphs

A **partial matching** μ matches edges to an incident vertex.

μ is **acyclic** if all sequences of matched pairs end at an unmatched vertex.

$$M_0 \triangleq \Sigma u \notin \mu, \quad M_1(c, d) \triangleq \Sigma(uv \notin \mu). (u \rightsquigarrow c) \times (v \rightsquigarrow d)$$

Example:



$$\mu \triangleq (v, vu), (w, wu), (x, xw)$$

$$M_0 \triangleq u$$

$$M_1(u, u) \triangleq (xu, [(x, xw), (w, wu)], [])$$

The Morse theorem in Cubical Agda

Use HIT to introduce homotopy type of a relation:

```
data |_| {c0 : Type} (c1 : c0 → c0 → Type) : Type where
  |_|_0 : c0 → | c1 |
  |_⇒_∃_|_1 : (x y : c0) → c1 x y → | x |_0 ≡ | y |_0
```

The Morse theorem in Cubical Agda

Use HIT to introduce homotopy type of a relation:

```
data |_| {c0 : Type} (c1 : c0 → c0 → Type) : Type where
  |_|_0 : c0 → | c1 |
  |_⇒_∃_|_1 : (x y : c0) → c1 x y → | x |_0 ≡ | y |_0
```

Example: for $M_0 \triangleq u$, $M_1(u, u) \triangleq (xu, [(x, xw), (w, wu)], [])$, it follows with simple pattern matching that $| M | \equiv S^1$ where $\text{base} : S^1$, $\text{loop} : \text{base} \equiv \text{base}$.

The Morse theorem in Cubical Agda

Use HIT to introduce homotopy type of a relation:

```
data |_| {c0 : Type} (c1 : c0 → c0 → Type) : Type where
  |_|_0 : c0 → | c1 |
  |_⇒_∃_|_1 : (x y : c0) → c1 x y → | x |_0 ≡ | y |_0
```

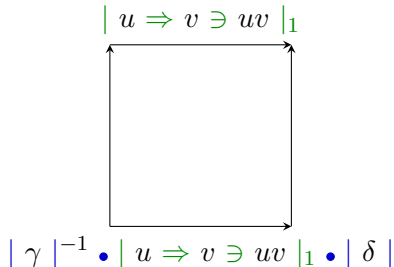
Example: for $M_0 \triangleq u$, $M_1(u, u) \triangleq (xu, [(x, xw), (w, wu)], [])$, it follows with simple pattern matching that $| M | \equiv S^1$ where $\text{base} : S^1$, $\text{loop} : \text{base} \equiv \text{base}$.

Morse theorem for graphs: $| G | \equiv | M |$

- establish maps back and forth, e.g., define $| M_1 | \rightarrow | E |$:
 $| c \Rightarrow d \ni (uv, \gamma, \delta) |_1 \mapsto | \gamma |^{-1} \bullet | u \Rightarrow v \ni uv |_1 \bullet | \delta |$
- show that these maps are mutually inverse.

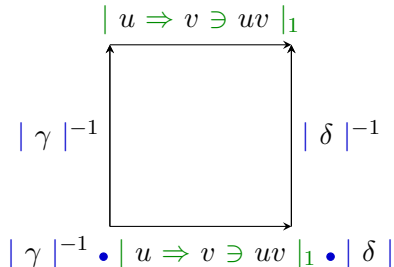
Proving the Morse Theorem

E.g., for any critical edge $| u \Rightarrow v \ni uv |_1$ show that mapping via $| E | \rightarrow | M_1 | \rightarrow | E |$ gives an edge homotopic to the original edge:



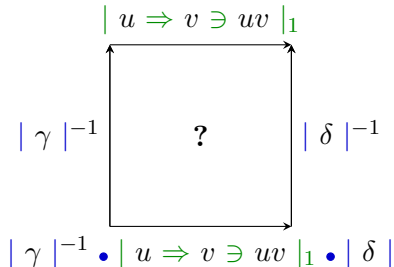
Proving the Morse Theorem

E.g., for any critical edge $| u \Rightarrow v \ni uv |_1$ show that mapping via $| E | \rightarrow | M_1 | \rightarrow | E |$ gives an edge homotopic to the original edge:



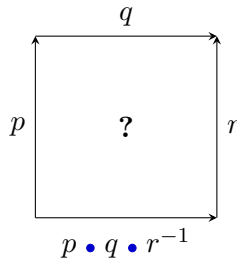
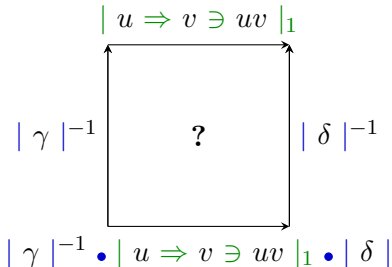
Proving the Morse Theorem

E.g., for any critical edge $| u \Rightarrow v \ni uv |_1$ show that mapping via $| E | \rightarrow | M_1 | \rightarrow | E |$ gives an edge homotopic to the original edge:



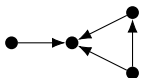
Proving the Morse Theorem

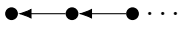
E.g., for any critical edge $| u \Rightarrow v \ni uv |_1$ show that mapping via $| E | \rightarrow | M_1 | \rightarrow | E |$ gives an edge homotopic to the original edge:



Establishing properties of graphs

Just have to define matching on a graph, then Morse theorem and univalence allows us to establish properties of graphs:

•  $\equiv \bigcirc$, so H_1 of this graph is \mathbb{Z} .

•  $\equiv \bullet$, so this infinite graph is contractible.

• ...

Next steps

- Formalise DMT for 2-dim complexes to compute topology of grayscale images¹

$$H_1(\pi) \cong \mathbb{Z}$$

- Implement full pipeline in [Cubical Agda](#): turn grayscale image into complex; compute APM; compute cohomology of reduced complex.
 - ▶ Linear algebra approach to computing invariants partially formalised², see how this connects with HoTT invariants³.
- Refine pipeline: filtered complexes for persistent homology, cellular sheaves, ...

¹Robins, Wood, Sheppard 2010, *Theory and Algorithms for Constructing Discrete Morse Complexes from Grayscale Digital Images*,

²Heras, Coquand, Mörtberg 2013, *Computing Persistent Homology Within Coq/SSReflect*

³work by Brunerie, Cavallo, Lamiaux, Ljungström, Mörtberg on “synthetic” cohomology (rings)

Conclusions

Summary

- Cubical Agda is a powerful tool—with a steep learning curve.
- Allows to directly reason about homotopy types.
- And at the same time it's a programming language!
Proofs can be erased to obtain performant Haskell programs.

Outlook

- Verified tool for topological data analysis on the horizon.
- Project can help bridge the gap between abstract and applied topology.
- **Cubical Agda** likely not the end of the story for higher-dimensional type theory, but gives insight into what reasoning in such a theory amounts to.

Outlook

- Verified tool for topological data analysis on the horizon.
- Project can help bridge the gap between abstract and applied topology.
- **Cubical Agda** likely not the end of the story for higher-dimensional type theory, but gives insight into what reasoning in such a theory amounts to.

Thank you for your attention!

Reasoning in Cubical Agda

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. We consider *cell contexts* capturing a single type.

Reasoning in Cubical Agda

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. We consider *cell contexts* capturing a single type.

Example: $\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$

$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

Reasoning in Cubical Agda

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. We consider *cell contexts* capturing a single type.

Example: $\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$

$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

Reasoning with **Kan filling**:

$$y \overset{\mathbf{t}_1}{\dashrightarrow} x$$

$$j \uparrow \downarrow i$$

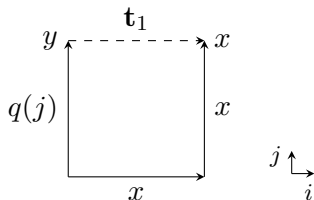
Reasoning in Cubical Agda

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. We consider *cell contexts* capturing a single type.

Example: $\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$

$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

Reasoning with **Kan filling**:



$\Gamma \mid i \vdash \mathbf{t}_1 : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto x]$ where

$\mathbf{t}_1 \triangleq \text{fill}^{\mathbf{0} \rightarrow \mathbf{1}} (j.[i = \mathbf{0} \mapsto q(j) \mid i = \mathbf{1} \mapsto x]) \ x$

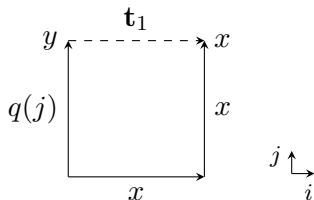
Reasoning in Cubical Agda

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. We consider *cell contexts* capturing a single type.

Example: $\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$

$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

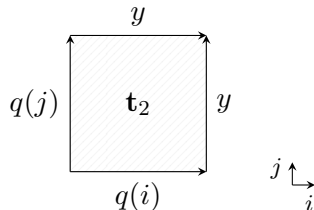
Reasoning with **Kan filling**:



$\Gamma \mid i \vdash \mathbf{t}_1 : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto x]$ where

$\mathbf{t}_1 \triangleq \text{fill}^{\mathbf{0} \mapsto \mathbf{1}} (j.[i = \mathbf{0} \mapsto q(j) \mid i = \mathbf{1} \mapsto x]) x$

Reasoning with **contortions**:



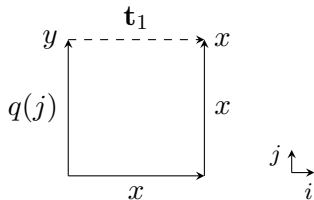
Reasoning in Cubical Agda

In cubical type theory, equalities between elements of a type are *paths*, i.e., maps from \mathbf{I}^n into that type. We consider *cell contexts* capturing a single type.

Example: $\Gamma \triangleq x : [], y : [], q(i) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto y]$

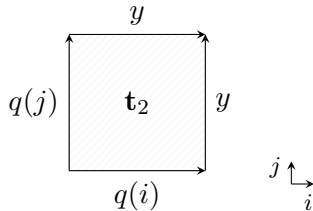
$$x \xrightarrow{q(i)} y \xrightarrow{i}$$

Reasoning with **Kan filling**:



$\Gamma \mid i \vdash \mathbf{t}_1 : [i = \mathbf{0} \mapsto y \mid i = \mathbf{1} \mapsto x]$ where
 $\mathbf{t}_1 \triangleq \text{fill}^{\mathbf{0} \rightarrow \mathbf{1}} (j.[i = \mathbf{0} \mapsto q(j) \mid i = \mathbf{1} \mapsto x]) \ x$

Reasoning with **contortions**:



$\Gamma \mid i, j \vdash \mathbf{t}_2 : \left[\begin{array}{l|l} i = \mathbf{0} \mapsto q(j) & i = \mathbf{1} \mapsto y \\ j = \mathbf{0} \mapsto q(i) & j = \mathbf{1} \mapsto y \end{array} \right]$
 where $\mathbf{t}_2 \triangleq q(i \vee j)$

Dedekind contortions

Cubical type theories differ in which contortions they support:

- *Cartesian* (**red**tt): degeneracies, symmetries $(i, j \vdash s(j, i))$ and diagonals $(i \vdash s(i, i))$.
- *Dedekind*: also connectives \wedge and \vee .
- *De Morgan* (**Cubical Agda**): also reversal \sim .

Dedekind contortions

Cubical type theories differ in which contortions they support:

- *Cartesian* (**red**tt): degeneracies, symmetries $(i, j \vdash s(j, i))$ and diagonals $(i \vdash s(i, i))$.
- *Dedekind*: also connectives \wedge and \vee .
- *De Morgan* (**Cubical Agda**): also reversal \sim .

Trade-off: more powerful contortion theory makes many Kan fillings superfluous, but number of contortions grows super-exponentially.

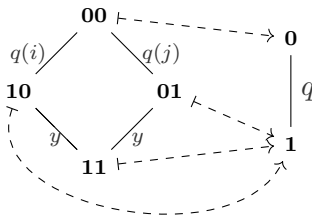
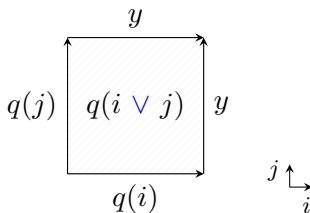
Dedekind contortions

Cubical type theories differ in which contortions they support:

- *Cartesian* (**redtt**): degeneracies, symmetries $(i, j \vdash s(j, i))$ and diagonals $(i \vdash s(i, i))$.
- *Dedekind*: also connectives \wedge and \vee .
- *De Morgan* (**Cubical Agda**): also reversal \sim .

Trade-off: more powerful contortion theory makes many Kan fillings superfluous, but number of contortions grows super-exponentially.

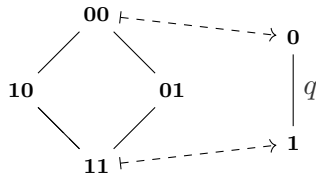
💡 Dedekind contortions from m - to n -cubes correspond to poset maps $\mathbf{I}^m \rightarrow \mathbf{I}^n$.



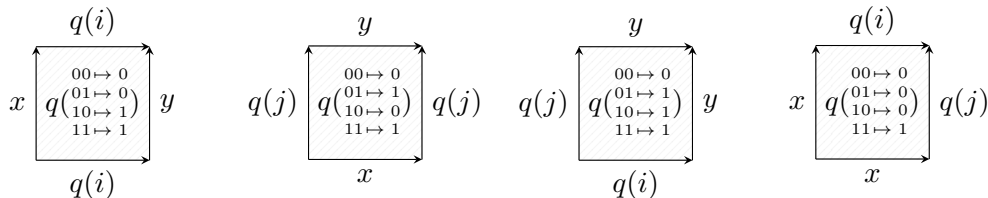
Capturing the contortion search space

Keep track of possible targets for any element:

$$q \begin{pmatrix} 00 \mapsto \{0\} \\ 01 \mapsto \{0, 1\} \\ 10 \mapsto \{0, 1\} \\ 11 \mapsto \{1\} \end{pmatrix}$$



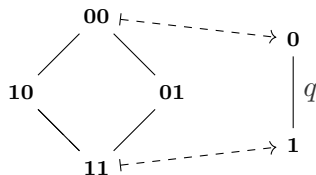
Captures four different contortions:



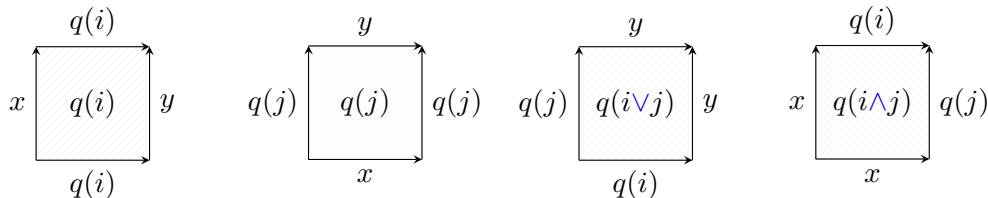
Capturing the contortion search space

Keep track of possible targets for any element:

$$q(\begin{matrix} 00 \mapsto \{0\} \\ 01 \mapsto \{0, 1\} \\ 10 \mapsto \{0, 1\} \\ 11 \mapsto \{1\} \end{matrix})$$



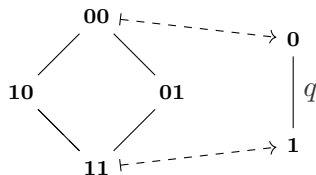
Captures four different contortions:



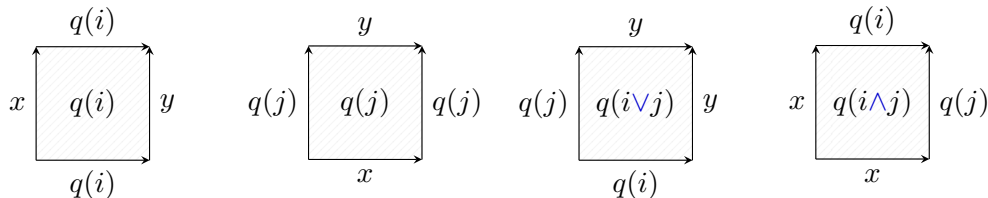
Capturing the contortion search space

Keep track of possible targets for any element:

$$q(\begin{matrix} 00 \mapsto \{0\} \\ 01 \mapsto \{0, 1\} \\ 10 \mapsto \{0, 1\} \\ 11 \mapsto \{1\} \end{matrix})$$





Captures four different contortions:



☞ Represent collection of contortions with *potential* poset map.

Contortion summary

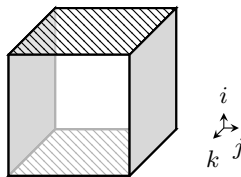
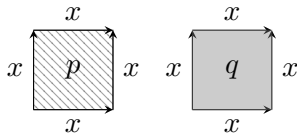
- Different contortion theories currently being considered, for proof search it's helpful to have available “truth-table” characterisation.
- Dedekind cubical type theory is such a theory; moreover, monotonicity allows more efficient proof search. 
- Contorting into dimension 1 shown NP-complete with reduction from SAT. 

Case study: Eckmann-Hilton in a cubical setting

Classical result: concatenation of 2-loops is commutative.

$$\Gamma \triangleq x : [], p(i, j), q(i, j) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x \mid j = \mathbf{0} \mapsto x \mid j = \mathbf{1} \mapsto x]$$

$$\Gamma \mid i, j, k \vdash ? : \left[\begin{array}{l} i = \mathbf{0} \mapsto p(j, k) \mid j = \mathbf{0} \mapsto q(i, k) \mid k = \mathbf{0} \mapsto x \\ i = \mathbf{1} \mapsto p(j, k) \mid j = \mathbf{1} \mapsto q(i, k) \mid k = \mathbf{1} \mapsto x \end{array} \right]$$

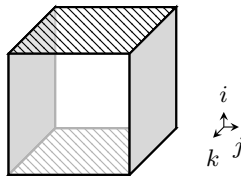
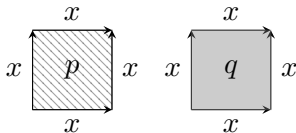


Case study: Eckmann-Hilton in a cubical setting

Classical result: concatenation of 2-loops is commutative.

$$\Gamma \triangleq x : [], p(i, j), q(i, j) : [i = \mathbf{0} \mapsto x \mid i = \mathbf{1} \mapsto x \mid j = \mathbf{0} \mapsto x \mid j = \mathbf{1} \mapsto x]$$

$$\Gamma \mid i, j, k \vdash ? : \left[\begin{array}{l} i = \mathbf{0} \mapsto p(j, k) \mid j = \mathbf{0} \mapsto q(i, k) \mid k = \mathbf{0} \mapsto x \\ i = \mathbf{1} \mapsto p(j, k) \mid j = \mathbf{1} \mapsto q(i, k) \mid k = \mathbf{1} \mapsto x \end{array} \right]$$



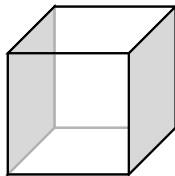
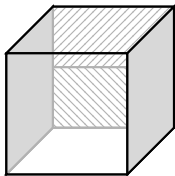
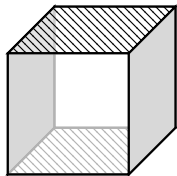
Construct as Kan filling of open 4-cube...

Proving Eckmann-Hilton automatically

```
> dedekind -f "examples/eckmannhilton.cube" -v
Cell context:
[("x",(Bdy 0 [])) , ("p",(Bdy 2 [ (1,I0) +> App "x" (1,[]) ... ])) , ...]
SOLVING problem 1...
(Bdy 3 [ (1,I0) +> App "p" (2,[[[1]],[[2]]]) , ... ])
SOLVE IN (4,I1) FOR (Bdy 3 [ ... ]) WITH OPEN SIDES []
INITIAL DOMAINS
((1,I0),[PApp "p" [([I0,I0,I0],[[I0,I0]]), ... ]])
((4,I0),[PApp "p" [([I0,I0,I0],[[I0,I0],[I0,I1],[I1,I0],[I1,I1]]), ...]])
...
SOLVED IN 158ms
λ i j k → hcomp (λ l → λ {
    (i = i0) → p j (k ∧ l) ; (j = i0) → q i k ; (k = i0) → x
    ; (i = i1) → p j (k ∧ l) ; (j = i1) → q i k ; (k = i1) → p j l
}) (q i k)
```

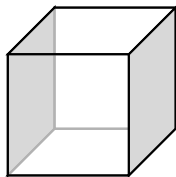
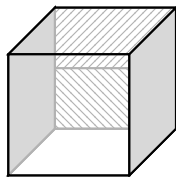
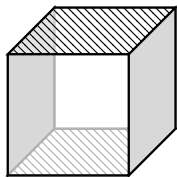
Making sense of the cubical Eckmann-Hilton argument


$$\Gamma \mid i, j, k \vdash \text{fill}^{0 \rightarrow 1} \left(l. \left[\begin{array}{c|c|c} i = 0 \mapsto p(j, k \vee l) & j = 0 \mapsto q(i, k) & k = 0 \mapsto x \\ i = 1 \mapsto p(j, k \vee l) & j = 1 \mapsto q(i, k) & k = 1 \mapsto p(j, l) \end{array} \right] \right) q(i, k)$$



Making sense of the cubical Eckmann-Hilton argument

$$\Gamma \mid i, j, k \vdash \text{fill}^{0 \rightarrow 1} \left(l. \left[\begin{array}{c|c|c} i = 0 \mapsto p(j, k \vee l) & j = 0 \mapsto q(i, k) & k = 0 \mapsto x \\ i = 1 \mapsto p(j, k \vee l) & j = 1 \mapsto q(i, k) & k = 1 \mapsto p(j, l) \end{array} \right] \right) q(i, k)$$



Proof can be turned into witness of $p \bullet q \equiv q \bullet p$ using lower-dimensional Kan filling, which can also be derived automatically. 

Contributions

- Self-contained language to talk about paths in a type in a cubical fashion.
- Studied **Kan filling** as a logical principle: shown undecidable; devised algorithm based on CSP that solves many common boundary problems.
- Studied **contortions** of Dedekind cubical type theory: subproblem shown to be NP-complete; devised algorithm to construct cubes with $\dim > 5$.
- Experimental implementation: <https://github.com/maxdore/dedekind>

Future work

- Incorporate solver into [Cubical Agda](#).
- Refine and explore new heuristics for constructing Kan fillings.
- Understand complexity of contortions better.
- Extend with other cubical features (heterogeneous equality and transports).
- Combine with automated reasoning for dependent type theory.

Future work

- Incorporate solver into [Cubical Agda](#).
- Refine and explore new heuristics for constructing Kan fillings.
- Understand complexity of contortions better.
- Extend with other cubical features (heterogeneous equality and transports).
- Combine with automated reasoning for dependent type theory.

Thank you for your attention!