Automating Reasoning in Cubical Type Theory

Prospects of Formal Mathematics
Trimester Program at HIM Bonn

28 June 2024

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

jww Evan Cavallo & Anders Mörtberg

Reasoning with Kan compositions

In cubical type theory, equalities over a type A are captured with paths, i.e., maps from $\mathbf{I}^n$ into A. Fix A in the following.

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

Reasoning with Kan compositions

In cubical type theory, equalities over a type A are captured with paths, i.e., maps from $\mathbf{I}^n$ into A. Fix A in the following.

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

$$x \xrightarrow{x} x \quad \overrightarrow{i}$$

Reasoning with Kan compositions

In cubical type theory, equalities over a type A are captured with paths, i.e., maps from $\mathbf{I}^n$ into A. Fix A in the following.

$$x : [] \mid i \vdash x : [i{=}0 \mapsto x \mid i{=}1 \mapsto x] \qquad x \xrightarrow{x} x \quad \overrightarrow{i}$$

Construct paths by *composing higher-dimensional open cubes.*

Reasoning with Kan compositions

In cubical type theory, equalities over a type A are captured with paths, i.e., maps from $\mathbf{I}^n$ into A. Fix A in the following.

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

$$x \xrightarrow{x} x \quad \overrightarrow{i}$$

Construct paths by *composing higher-dimensional open cubes.*

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

$$x : [], y : [], q(i) : [i{=}0 \mapsto x \mid i{=}1 \mapsto y] \mid$$
$$i \vdash ? : [i{=}0 \mapsto y \mid i{=}1 \mapsto x]$$

Reasoning with Kan compositions

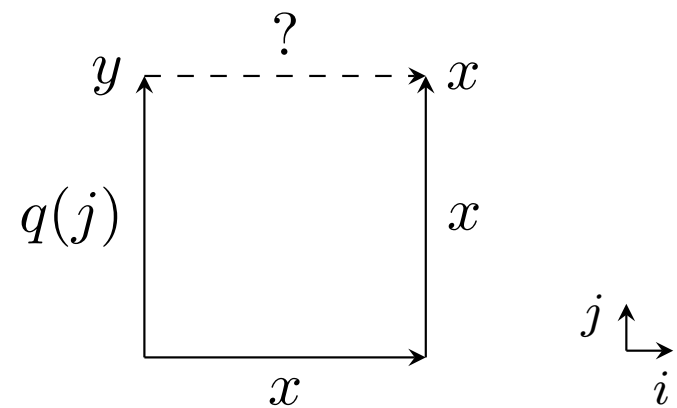
In cubical type theory, equalities over a type A are captured with paths, i.e., maps from $\mathbf{I}^n$ into A. Fix A in the following.

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

$$x \xrightarrow{\;x\;} x \quad \overrightarrow{i}$$

Construct paths by *composing higher-dimensional open cubes.*

$$x : [], y : [], q(i) : [i{=}0 \mapsto x \mid i{=}1 \mapsto y] \mid i \vdash ? : [i{=}0 \mapsto y \mid i{=}1 \mapsto x]$$



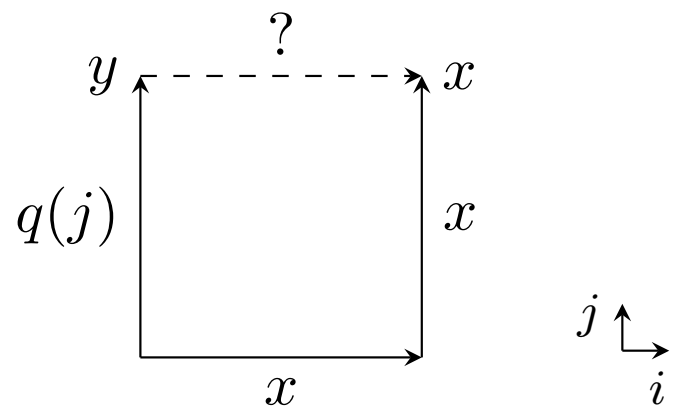
Reasoning with Kan compositions

In cubical type theory, equalities over a type A are captured with paths, i.e., maps from $\mathbf{I}^n$ into A. Fix A in the following.

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

Construct paths by *composing higher-dimensional open cubes.*

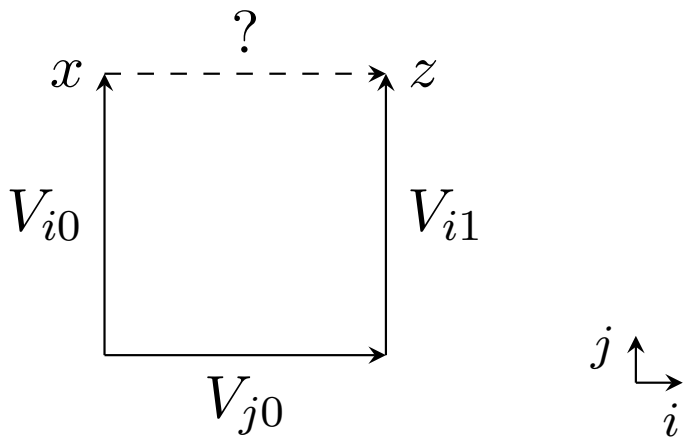
$$x : [], y : [], q(i) : [i{=}0 \mapsto x \mid i{=}1 \mapsto y] \mid i \vdash ? : [i{=}0 \mapsto y \mid i{=}1 \mapsto x]$$



Solution: $? = \mathsf{hcomp}\ (j.[i{=}0 \mapsto q(j) \mid i{=}1 \mapsto x])\ x$

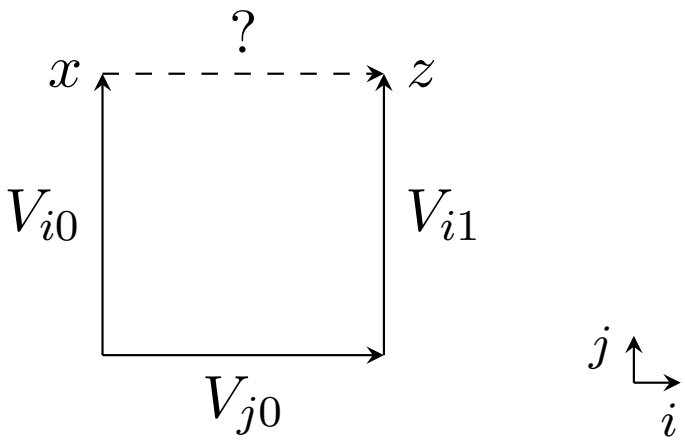
Finding open cubes with constraint solving

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



Finding open cubes with constraint solving

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



Domains for variables

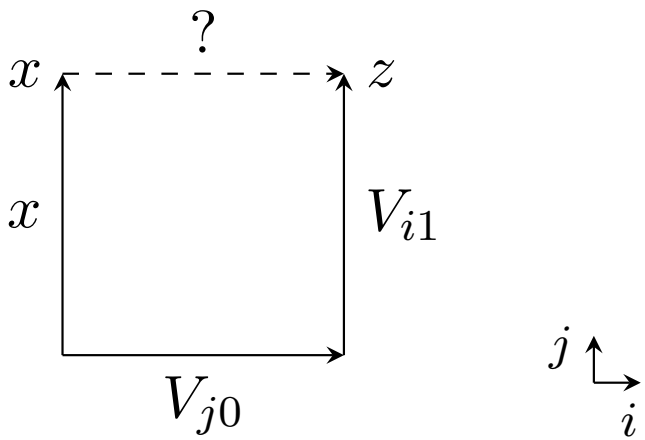
- $D_{i0} = \{x\}$
- $D_{i1} = \{z, r(j)\}$
- $D_{j0} = \{x, y, z, q(i), r(i)\}$

subject to constraints

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

Finding open cubes with constraint solving

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



Domains for variables

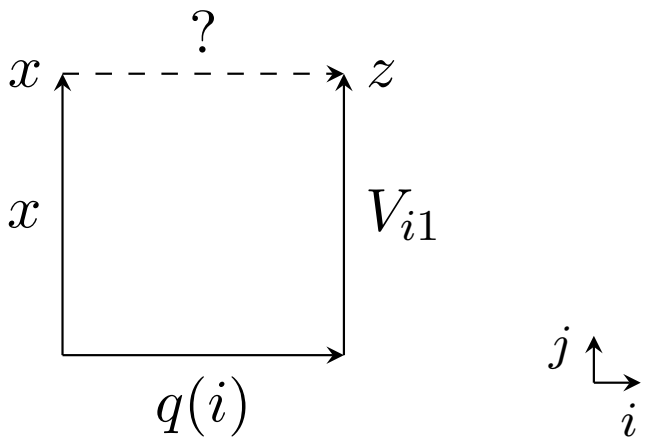
- $D_{i0} = \{x\}$
- $D_{i1} = \{z, r(j)\}$
- $D_{j0} = \{x, \cancel{y}, \cancel{z}, q(i), \cancel{r(i)}\}$

subject to constraints

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

Finding open cubes with constraint solving

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



Domains for variables

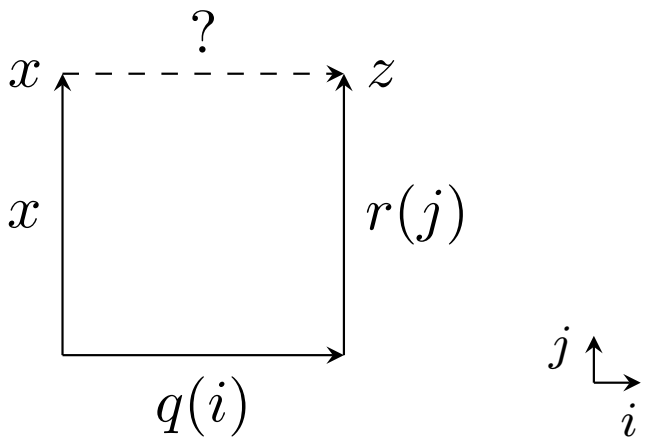
- $D_{i0} = \{x\}$
- $D_{i1} = \{\not{z}, r(j)\}$
- $D_{j0} = \{x, \not{y}, \not{z}, q(i), \not{r(i)}\}$

subject to constraints

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

Finding open cubes with constraint solving

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



Domains for variables

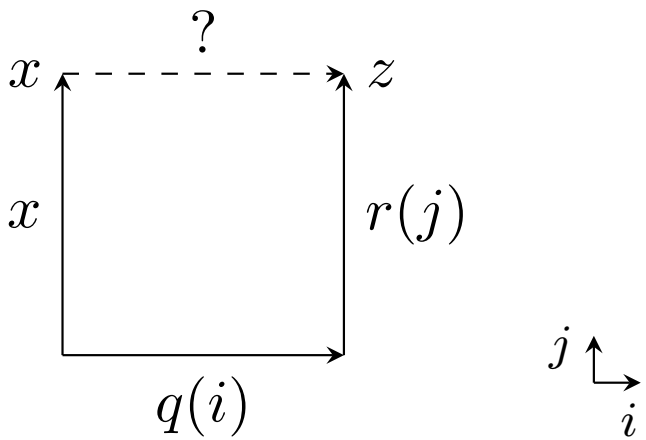
- $D_{i0} = \{x\}$
- $D_{i1} = \{\not{z}, r(j)\}$
- $D_{j0} = \{\not{x}, \not{y}, \not{z}, q(i), \not{r(i)}\}$

subject to constraints

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

Finding open cubes with constraint solving

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



Domains for variables

- $D_{i0} = \{x\}$
- $D_{i1} = \{\not{z}, r(j)\}$
- $D_{j0} = \{\not{x}, \not{y}, \not{z}, q(i), \not{r(i)}\}$

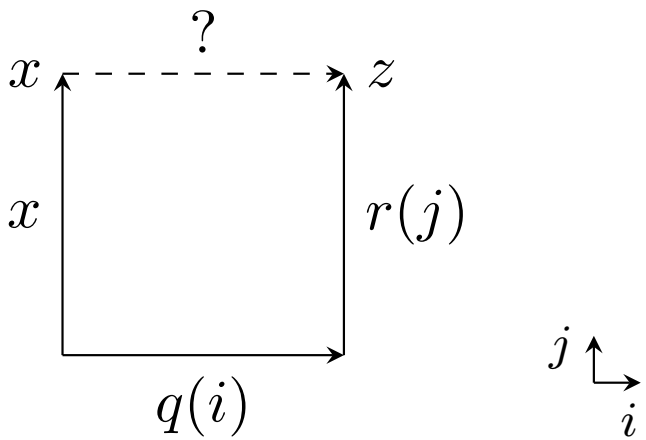
subject to constraints

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

Solution: $? = \mathsf{hcomp}\ (j.[i{=}0 \mapsto x \mid i{=}1 \mapsto r(j)])\ q(i)$

Finding open cubes with constraint solving

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



Domains for variables

- $D_{i0} = \{x\}$
- $D_{i1} = \{\not{z}, r(j)\}$
- $D_{j0} = \{\not{x}, \not{y}, \not{z}, q(i), \not{r(i)}\}$

subject to constraints

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

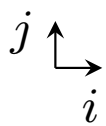
Solution: $? = \mathsf{hcomp}\ (j.[i{=}0 \mapsto x \mid i{=}1 \mapsto r(j)])\ q(i)$

Find Kan compositions using finite domain constraint solving.

Searching for nested Kan compositions

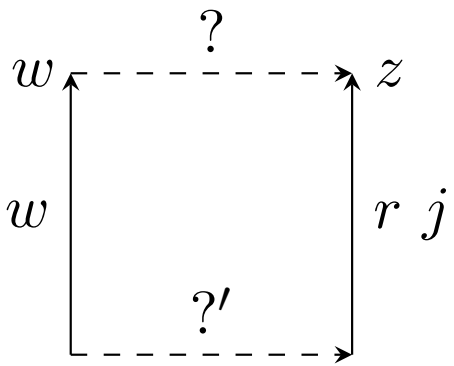
$p(i) : [i{=}0 \mapsto w \mid i{=}1 \mapsto x], q(i) : [i{=}0 \mapsto x \mid i{=}1 \mapsto y], r(i) : [i{=}0 \mapsto y \mid i{=}1 \mapsto z]$
$i \vdash ? : [i{=}0 \mapsto w \mid i{=}1 \mapsto z]$

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

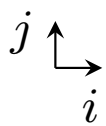


Searching for nested Kan compositions

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

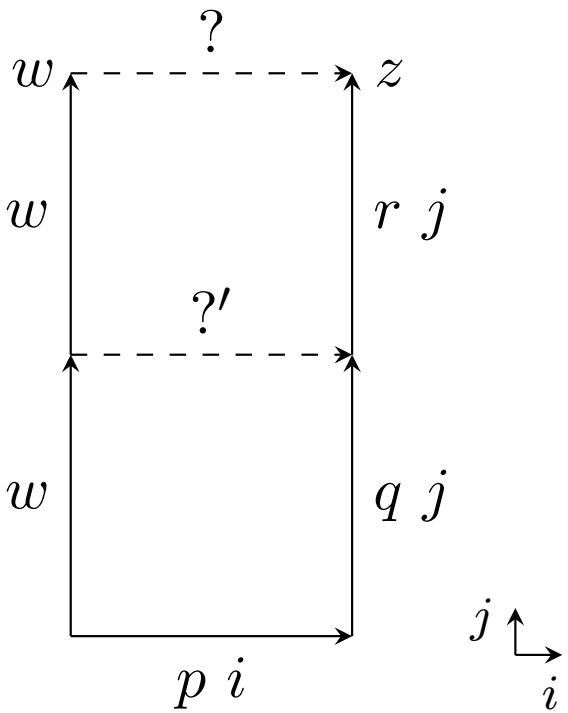


$$? = \mathsf{hcomp}\ (j.[i{=}0 \mapsto w \mid i{=}1 \mapsto r(j)])\ ?'$$



Searching for nested Kan compositions

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

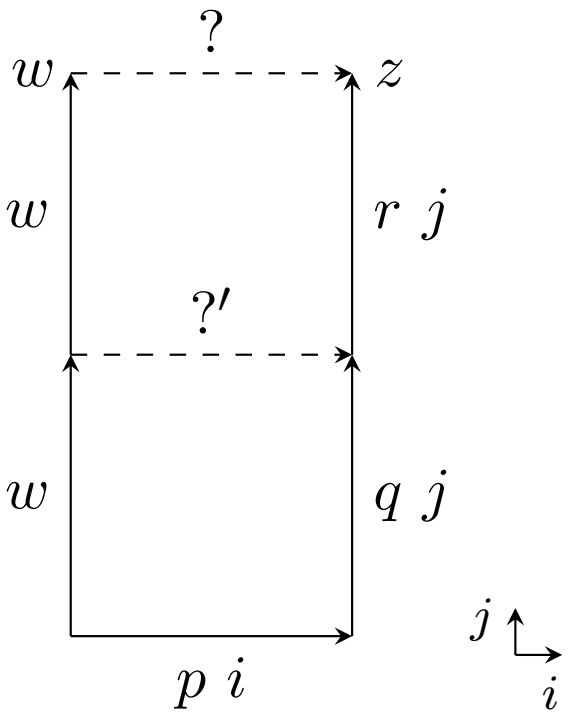


$? = \mathsf{hcomp}\ (j.[i{=}0 \mapsto w \mid i{=}1 \mapsto r(j)])\ ?'$

$?' = \mathsf{hcomp}\ (j.[i{=}0 \mapsto w \mid i{=}1 \mapsto q(j)])\ p(i)$

Searching for nested Kan compositions

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



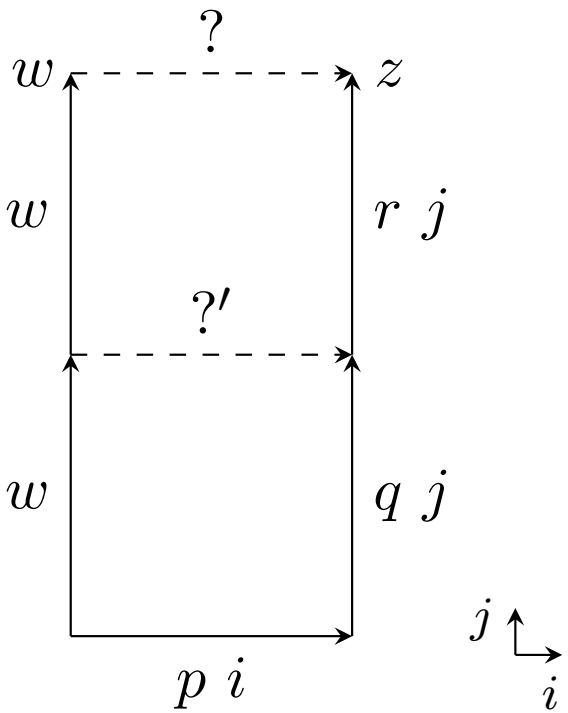
$? = \mathsf{hcomp}\ (j.[i{=}0 \mapsto w \mid i{=}1 \mapsto r(j)])\ ?'$

$?' = \mathsf{hcomp}\ (j.[i{=}0 \mapsto w \mid i{=}1 \mapsto q(j)])\ p(i)$

Use as many simple terms as possible, fill the remaining sides with Kan compositions afterwards.

Searching for nested Kan compositions

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



$? = \mathsf{hcomp}\ (j.[i{=}0 \mapsto w \mid i{=}1 \mapsto r(j)])\ ?'$

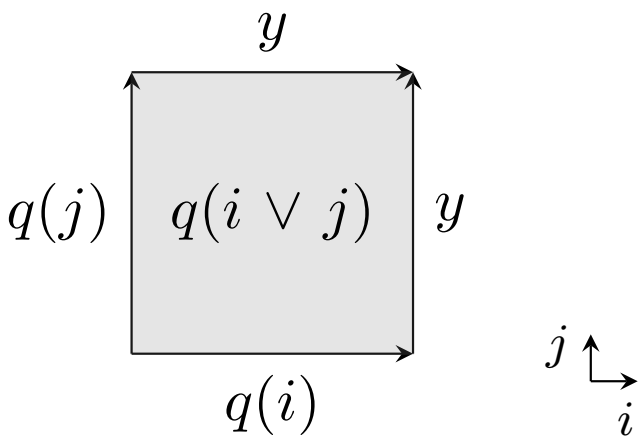
$?' = \mathsf{hcomp}\ (j.[i{=}0 \mapsto w \mid i{=}1 \mapsto q(j)])\ p(i)$

Use as many simple terms as possible, fill the remaining sides with Kan compositions afterwards.

Constructing cubes with contortions

CTTs often come with operations on the interval variables.

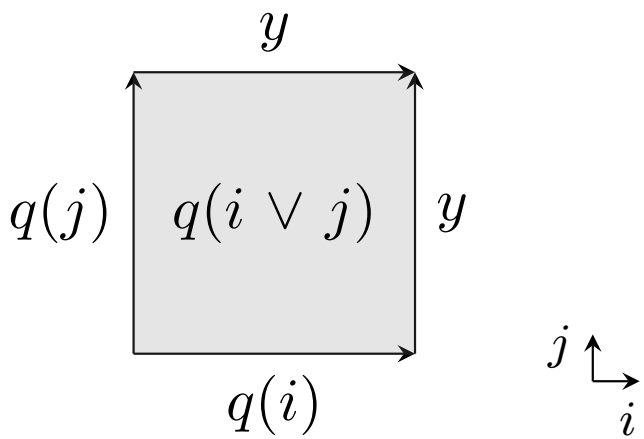
$$i, j \vdash q(i \vee j) : [i{=}0 \mapsto q(j) \mid i{=}1 \mapsto y \mid j{=}0 \mapsto q(i) \mid j{=}1 \mapsto y]$$



Constructing cubes with contortions

CTTs often come with operations on the interval variables.

$$i, j \vdash q(i \vee j) : [i{=}0 \mapsto q(j) \mid i{=}1 \mapsto y \mid j{=}0 \mapsto q(i) \mid j{=}1 \mapsto y]$$

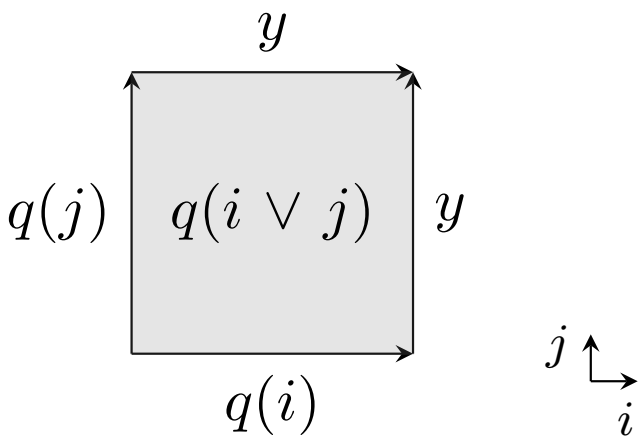


Trade-off: contortions makes large class of Kan compositions superfluous, but makes search for contorted cubes non-trivial—*Dedekind* contortions $\wedge$ and $\vee$ yield $D(n)$-th ways to turn 1-cube into an n-cube (e.g., $D(6) = 7\,828\,354$)

Constructing cubes with contortions

CTTs often come with operations on the interval variables.

$$i, j \vdash q(i \vee j) : [i{=}0 \mapsto q(j) \mid i{=}1 \mapsto y \mid j{=}0 \mapsto q(i) \mid j{=}1 \mapsto y]$$



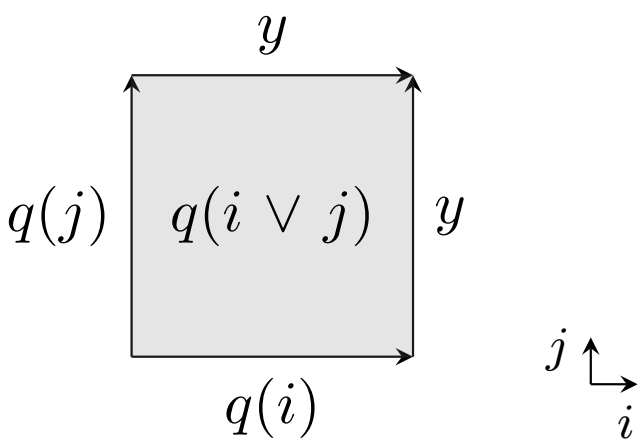
Trade-off: contortions makes large class of Kan compositions superfluous, but makes search for contorted cubes non-trivial—*Dedekind* contortions $\wedge$ and $\vee$ yield $D(n)$-th ways to turn 1-cube into an n-cube (e.g., $D(6) = 7\,828\,354$)

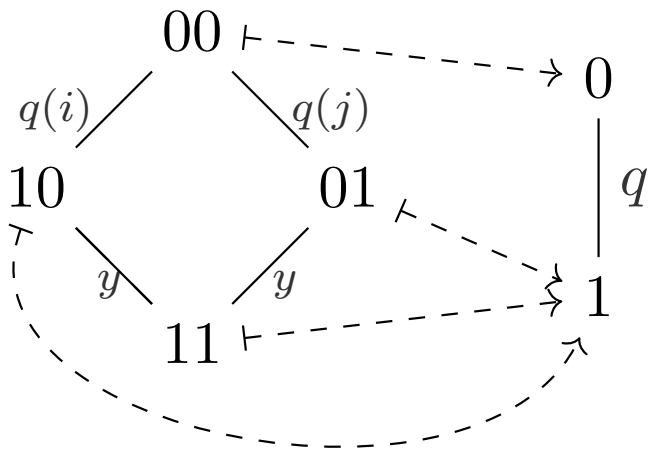
$\rightarrow$ Use characterisation of Dedekind contortions as poset maps to represent search space concisely.

Constructing cubes with contortions

CTTs often come with operations on the interval variables.

$$i, j \vdash q(i \vee j) : [i{=}0 \mapsto q(j) \mid i{=}1 \mapsto y \mid j{=}0 \mapsto q(i) \mid j{=}1 \mapsto y]$$





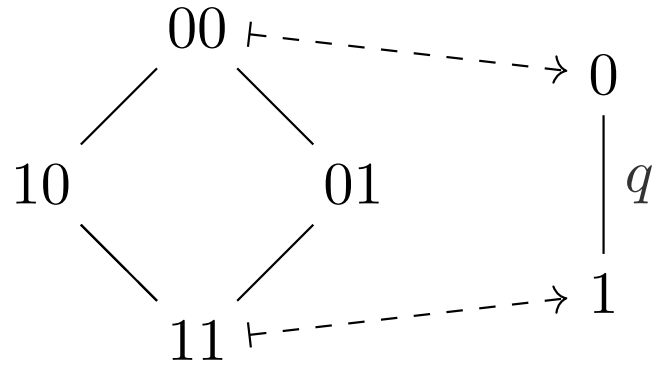
Trade-off: contortions makes large class of Kan compositions superfluous, but makes search for contorted cubes non-trivial—*Dedekind* contortions $\wedge$ and $\vee$ yield $D(n)$-th ways to turn 1-cube into an n-cube (e.g., $D(6) = 7\,828\,354$)

$\rightarrow$ Use characterisation of Dedekind contortions as poset maps to represent search space concisely.

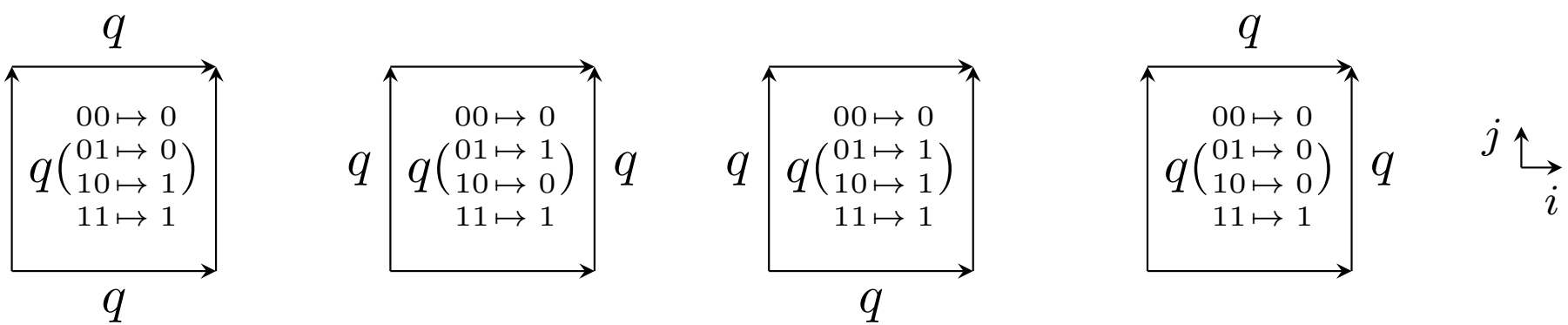
Collections of contortions as potential poset maps

Save for any element of the domain all possible targets.

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



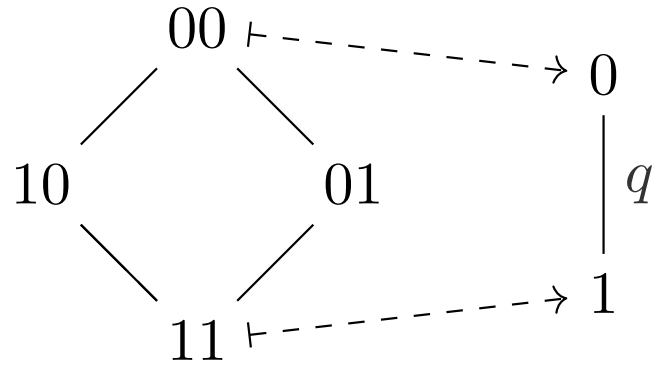
Captures four different contortions:



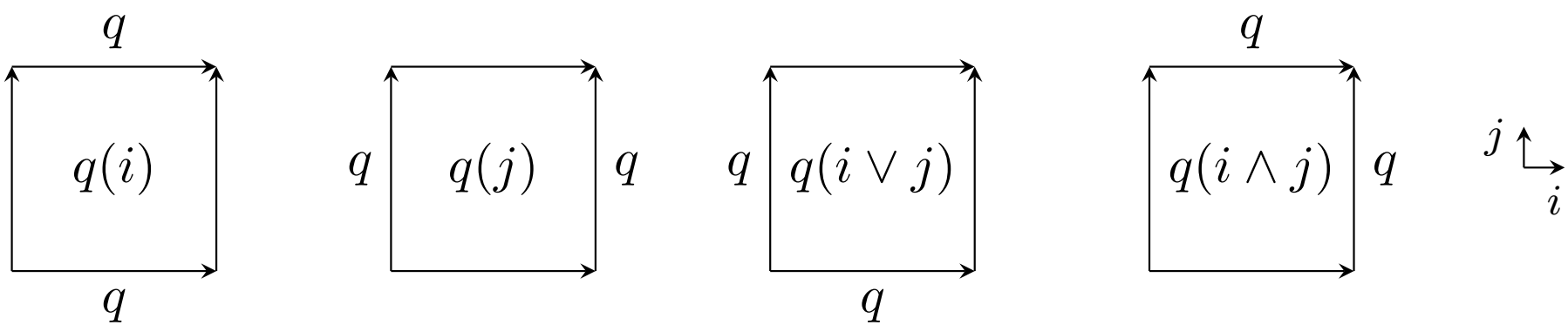
Collections of contortions as potential poset maps

Save for any element of the domain all possible targets.

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



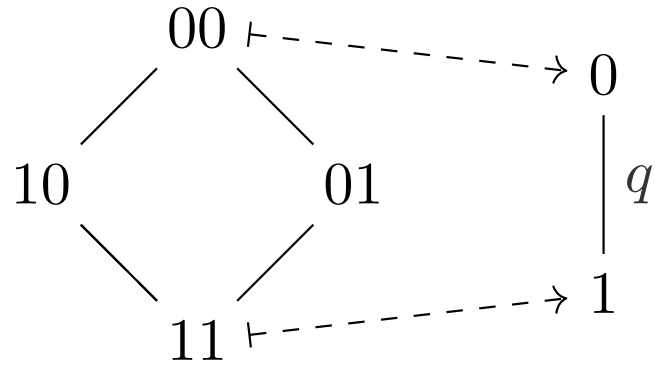
Captures four different contortions:



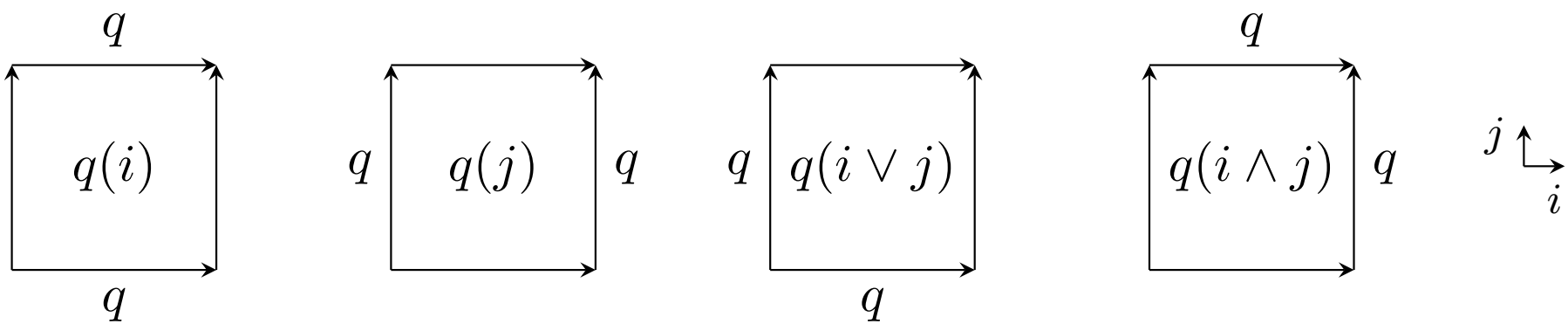
Collections of contortions as potential poset maps

Save for any element of the domain all possible targets.

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



Captures four different contortions:

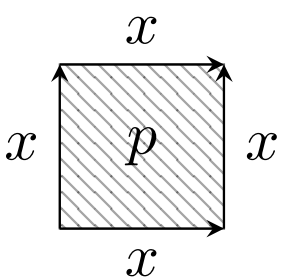


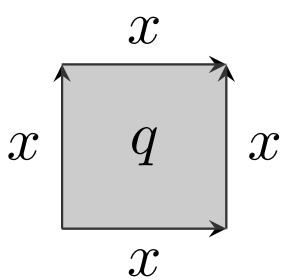
Construct contortions gradually by restricting poset map.

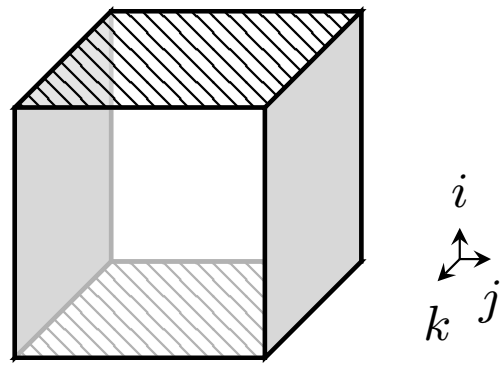
Eckmann-Hilton in a cubical setting

Classical result: concatenation of 2-loops is commutative.

$$x : [], p(i,j), q(i,j) : [i{=}0 \mapsto x \mid i{=}1 \mapsto x \mid j{=}0 \mapsto x \mid j{=}1 \mapsto x] \quad \mid$$
$$i,j,k \vdash ? : [\begin{array}{lll} i{=}0 \mapsto p(j,k) \mid & j{=}0 \mapsto q(i,k) \mid & k{=}0 \mapsto x \\ i{=}1 \mapsto p(j,k) \mid & j{=}1 \mapsto q(i,k) \mid & k{=}1 \mapsto x \end{array}]$$



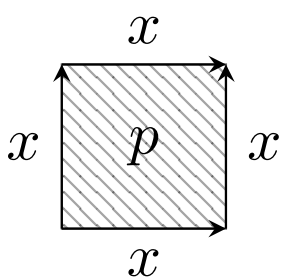


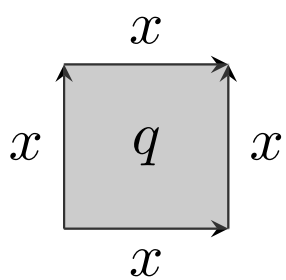


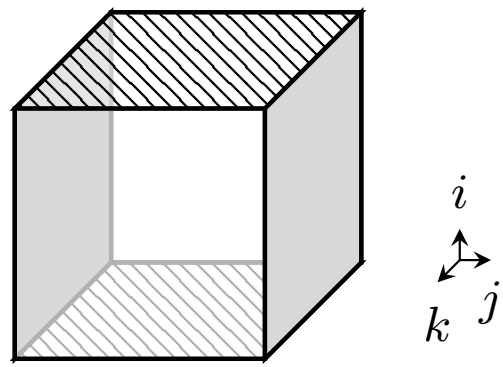
Eckmann-Hilton in a cubical setting

Classical result: concatenation of 2-loops is commutative.

$$x : [], p(i,j), q(i,j) : [i{=}0 \mapsto x \mid i{=}1 \mapsto x \mid j{=}0 \mapsto x \mid j{=}1 \mapsto x] \quad \mid$$
$$i,j,k \vdash ? : [\begin{array}{lll} i{=}0 \mapsto p(j,k) \mid & j{=}0 \mapsto q(i,k) \mid & k{=}0 \mapsto x \\ i{=}1 \mapsto p(j,k) \mid & j{=}1 \mapsto q(i,k) \mid & k{=}1 \mapsto x \end{array}]$$



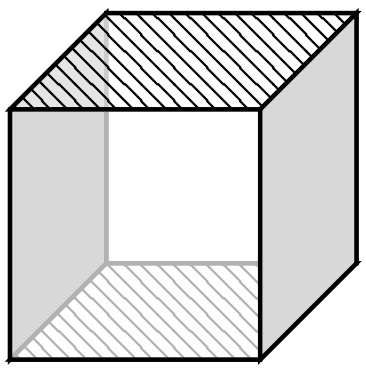


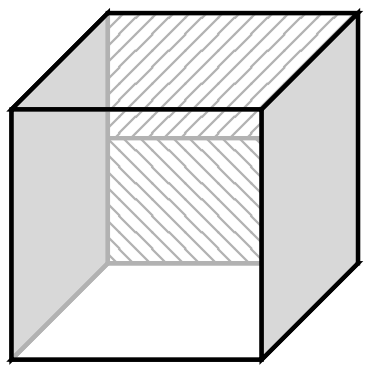


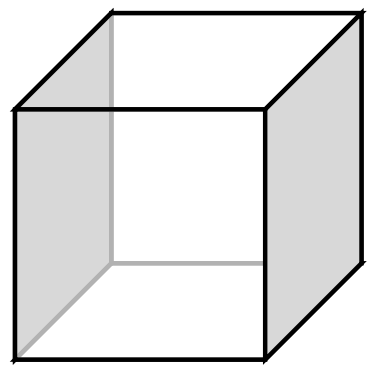
→ Construct as Kan composition of open 4-cube.

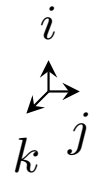
The cubical Eckmann-Hilton argument

$$
\begin{array}{lll}
 & i{=}0 \mapsto p(j, k \vee l) & \mid\ i{=}1 \mapsto p(j, k \vee l) \\
i, j, k \vdash \mathsf{hcomp}\ (l.[\ & j{=}0 \mapsto q(i, k) & \mid\ j{=}1 \mapsto q(i, k) \qquad])\ q(i, k) \\
 & k{=}0 \mapsto x & \mid\ k{=}1 \mapsto p(j, l)
\end{array}
$$









Summary

We studied the problem of constructing certain cubes using

- *Contortions*: operations on the interval variables giving basic shapes (decidable but huge search space),
- *Kan compositions*: gives missing side of open cube (undecidable).

Summary

We studied the problem of constructing certain cubes using

- *Contortions*: operations on the interval variables giving basic shapes (decidable but huge search space),
- *Kan compositions*: gives missing side of open cube (undecidable).

Our algorithm is based on

1. finite domain constraint solving,
2. exploring nested cubes in a "breadth-first" way,
3. representing collections of contortions as poset maps.

Future work

- Incorporate solver into Cubical Agda.
- Refine heuristics.
- Automate other cubical reasoning principles:
  - paths between types (heterogeneous equality),
  - transporting along paths (transp),
  - instances of univalence (Glue).
- Combine with synthesis of dependent type theory.

Future work

- Incorporate solver into Cubical Agda.
- Refine heuristics.
- Automate other cubical reasoning principles:
  - paths between types (heterogeneous equality),
  - transporting along paths (transp),
  - instances of univalence (Glue).
- Combine with synthesis of dependent type theory.

Implementation: `https://github.com/maxdore/dedekind`

Description: `https://arxiv.org/abs/2402.12169`
"Automating Boundary Filling in Cubical Agda"

Future work

- Incorporate solver into Cubical Agda.
- Refine heuristics.
- Automate other cubical reasoning principles:
  - paths between types (heterogeneous equality),
  - transporting along paths (transp),
  - instances of univalence (Glue).
- Combine with synthesis of dependent type theory.

Implementation: `https://github.com/maxdore/dedekind`

Description: `https://arxiv.org/abs/2402.12169`
"Automating Boundary Filling in Cubical Agda"

Thank you for your attention!

Eckmann-Hilton as its commonly known

```
Sq→Comp : PathP (λ j → q j ≡ q j) p p → p • q ≡ q • p
Sq→Comp  i j = hcomp (λ k → λ {
  (i = i0) → hcomp (λ l → λ {
    (j = i0) → x ; (k = i0) → q (j ∧ l) ; (j = i1) →  l k ;
    (k = i1) → hfill (λ m → λ { (j = i0) → x ; (j = i1) → q m })
    (p (j ∧ k)) ;
  (i = i1) → hfill (λ l → λ { (j = i0) → x ; (j = i1) → p l }) (inS
  (j = i0) → x ; (j = i1) → p k })
  (q j)

EckmannHilton' : p • q ≡ q • p
EckmannHilton' = Sq→Comp p q EckmannHilton
```