Automating Kan composition

WG6 kick-off meeting: Syntax and Semantics of Type Theories

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

20 May 2022

A tactic for Cubical Agda

PhD project:

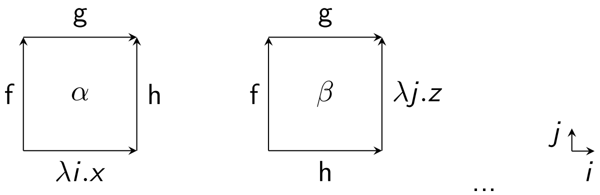
- Verify some results in computational topology: Use cubical type theory to study homotopy types of simplicial complexes.
- Central in many proofs: Use built-in Kan composition to show contractibility of certain types.

Steep learning curve when using hcomp's...

Goal: Automatic procedure to construct open cubes whose lids prove a goal under question.

Simplices to cubes

Example: We can map a triangle $\alpha : g \circ f \Rightarrow h$ between $f : x \to y$, $g : y \to z$ and $h : x \to z$ into cubical sets in different ways:



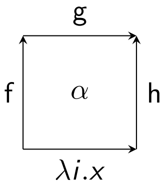
How can we turn α into β?

Syntax

We look at a fragment of Cubical Agda with PathP and $\wedge$, $\vee$.
Given infinite set of dimensions $D = \{i, j, ...\}$, endpoints $0_{\mathcal{I}}$, $1_{\mathcal{I}}$:

$$
\begin{array}{rcll}
\Gamma & ::= & () \mid a : \Phi\ ,\ \Gamma & (\textit{contexts}) \\
r, s & ::= & i \mid (r \vee s) \mid (r \wedge s) & (\textit{formulas}) \\
t, u, v & ::= & a \mid \lambda i.t \mid t\ r & (\textit{terms}) \\
\Phi & ::= & \bullet \mid \text{PathP}\ (\lambda i.\Phi)\ u\ v & (\textit{shapes})
\end{array}
$$

Example:
$x : \bullet, y : \bullet, z : \bullet, f : \text{PathP} \bullet x\ y,$
$g : \text{PathP} \bullet y\ z, h : \text{PathP} \bullet x\ z$
$\alpha : \text{PathP}\ (\lambda j.\text{PathP} \bullet (f\ j)\ (h\ j))\ \lambda i.x\ g$

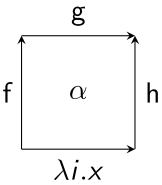


Syntax

We look at a fragment of Cubical Agda with PathP and $\wedge$, $\vee$.
Given infinite set of dimensions $D = \{i, j, ...\}$, endpoints $0_{\mathcal{I}}$, $1_{\mathcal{I}}$:

$$\begin{aligned}
\Gamma &::= () \mid a : \Phi \ , \ \Gamma & (\textit{contexts}) \\
r, s &::= i \mid (r \vee s) \mid (r \wedge s) & (\textit{formulas}) \\
t, u, v &::= a \mid \lambda i.t \mid t \ r & (\textit{terms}) \\
\Phi &::= \bullet \mid \mathrm{PathP}\ (\lambda i.\Phi)\ u\ v & (\textit{shapes})
\end{aligned}$$

Example:
$x : \bullet, y : \bullet, z : \bullet, f : \mathrm{PathP} \bullet x\ y,$
$g : \mathrm{PathP} \bullet y\ z, h : \mathrm{PathP} \bullet x\ z$
$\alpha : \mathrm{PathP}\ (\lambda j.\mathrm{PathP} \bullet (f\ j)\ (h\ j))\ \lambda i.x\ g$



Given $t : \Phi$, we will denote with $t|_{k=e}$, $\Phi|_{k=e}$ its $(k+e)$-th face.
Example: $\alpha|_{1=0_{\mathcal{I}}} = \lambda i.x$, $\alpha|_{2=1_{\mathcal{I}}} = h$.

An algorithm for filling cubes

Input: Γ a context with faces of some type, n-dimensional cube Φ
Output: A term $t : \Phi$ formed over Γ with abs, app and hcomp.

Step 1: Make all $(a : \Psi)$ in Γ with $\dim(\Psi) = m$ to n-cubes:

$$\lambda i_1 \ldots \lambda i_n . a \; r_1 \; \ldots \; r_m \text{ where } r_{\_} \in FreeDL(i_1, \ldots, i_n)$$

Let S be the collection of all degeneracies for terms in Γ.
If for some $t \in S$ we have $t : \Phi$, return t.

An algorithm for filling cubes

Input: Γ a context with faces of some type, n-dimensional cube Φ
Output: A term $t : \Phi$ formed over Γ with abs, app and hcomp.

Step 1: Make all $(a : \Psi)$ in Γ with $\dim(\Psi) = m$ to n-cubes:

$$\lambda i_1 ... \lambda i_n . a \; r_1 \; ... \; r_m \text{ where } r_\_ \in FreeDL(i_1, ..., i_n)$$

Let S be the collection of all degeneracies for terms in Γ.
If for some $t \in S$ we have $t : \Phi$, return t.

To fill the 2-dimensional square β from above, we generate

$$\begin{aligned} S = \{ & \lambda i.\lambda j.x, \lambda i.\lambda j.y, \lambda i.\lambda j.z, \\ & \lambda i.\lambda j.f \; i, \lambda i.\lambda j.f \; j, \lambda i.\lambda j.f \; (i \vee j), \lambda i.\lambda j.f \; (i \wedge j), ..., \\ & \lambda i.\lambda j.\alpha \; i \; i, \lambda i.\lambda j.\alpha \; i \; j, \lambda i.\lambda j.\alpha \; i \; (i \vee j), ..., \lambda i.\lambda j.\alpha \; (i \wedge j) \; (i \wedge j) \} \end{aligned}$$

Constructing an open $(n+1)$-dimensional cube

Step 2: Solve constraint satisfaction problem.

- Variables X_B and $X_{i_k,0_\mathcal{I}}$, $X_{i_k,1_\mathcal{I}}$ for $1 \leq k \leq n$.
- Domains $D_B = S$ and $D_{i_k,e} = \{t \in S : t|_{n=1_\mathcal{I}} = \Phi|_{k=e}\}$ for $1 \leq k \leq n$, $e \in \{0_\mathcal{I}, 1_\mathcal{I}\}$.
- Constraints
  - $X_{i_k,e}|_{n=0_\mathcal{I}} = X_B|_{k=e}$ for $k = 1, ..., n$, $e \in \{0_\mathcal{I}, 1_\mathcal{I}\}$.
  - $X_{i_k,e}|_{k+e=e'} = X_{i_l,e'}|_{l-e'=e}$ for $1 \leq k < l \leq n$, $e, e' \in \{0_\mathcal{I}, 1_\mathcal{I}\}$.

If solution exists, return *hcomp* $(X_{i_1,0_\mathcal{I}}, X_{i_1,1_\mathcal{I}}, ..., X_{i_n,0_\mathcal{I}}, X_{i_n,1_\mathcal{I}})$ X_B

Constructing an open $(n+1)$-dimensional cube

Step 2: Solve constraint satisfaction problem.

- Variables X_B and $X_{i_k,0_\mathcal{I}}$, $X_{i_k,1_\mathcal{I}}$ for $1 \leq k \leq n$.
- Domains $D_B = S$ and $D_{i_k,e} = \{t \in S : t|_{n=1_\mathcal{I}} = \Phi|_{k=e}\}$ for $1 \leq k \leq n$, $e \in \{0_\mathcal{I}, 1_\mathcal{I}\}$.
- Constraints
  - $X_{i_k,e}|_{n=0_\mathcal{I}} = X_B|_{k=e}$ for $k = 1, ..., n$, $e \in \{0_\mathcal{I}, 1_\mathcal{I}\}$.
  - $X_{i_k,e}|_{k+e=e'} = X_{i_l,e'}|_{l-e'=e}$ for $1 \leq k < l \leq n$, $e, e' \in \{0_\mathcal{I}, 1_\mathcal{I}\}$.

If solution exists, return *hcomp* $(X_{i_1,0_\mathcal{I}}, X_{i_1,1_\mathcal{I}}, ..., X_{i_n,0_\mathcal{I}}, X_{i_n,1_\mathcal{I}})\ X_B$

For an open 3-cube, constrain vars $X_B, X_{i,0_\mathcal{I}}, X_{i,1_\mathcal{I}}, X_{j,0_\mathcal{I}}, X_{i,1_\mathcal{I}}$:

$$X_{i_1,0_\mathcal{I}}|_{2=0_\mathcal{I}} = X_B|_{1=0_\mathcal{I}} \qquad X_{i_1,1_\mathcal{I}}|_{2=0_\mathcal{I}} = X_B|_{1=1_\mathcal{I}}$$

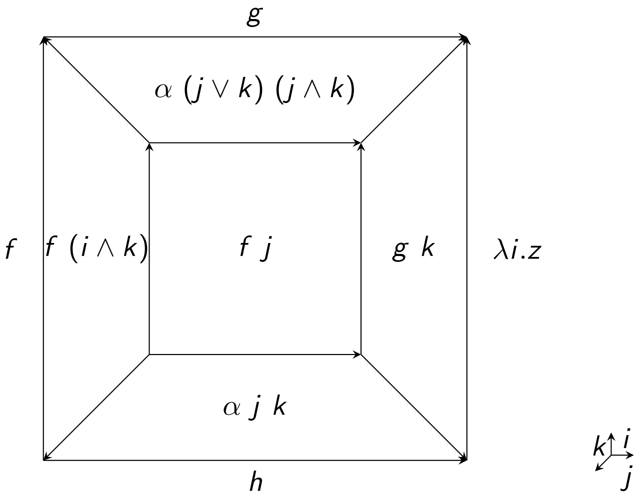
$$X_{i_2,0_\mathcal{I}}|_{2=0_\mathcal{I}} = X_B|_{2=0_\mathcal{I}} \qquad X_{i_2,1_\mathcal{I}}|_{2=0_\mathcal{I}} = X_B|_{2=1_\mathcal{I}}$$

$$X_{i_1,0_\mathcal{I}}|_{1=0_\mathcal{I}} = X_{i_2,0_\mathcal{I}}|_{2=0_\mathcal{I}} \qquad X_{i_1,0_\mathcal{I}}|_{1=1_\mathcal{I}} = X_{i_2,1_\mathcal{I}}|_{1=0_\mathcal{I}}$$

$$X_{i_1,1_\mathcal{I}}|_{2=0_\mathcal{I}} = X_{i_2,0_\mathcal{I}}|_{2=1_\mathcal{I}} \qquad X_{i_1,1_\mathcal{I}}|_{2=0_\mathcal{I}} = X_{i_2,1_\mathcal{I}}|_{1=1_\mathcal{I}}$$

Demo: finite-domain constraint solver to derive hcomps. Current implementation based on [Schrijvers et al., 2009].

```
https://github.com/maxdore/csolver/
```


g
$\alpha\ (j \vee k)\ (j \wedge k)$
f
$f\ (i \wedge k)$
$f\ j$
$g\ k$
$\lambda i.z$
$\alpha\ j\ k$
h
k
i
j

Geometric proof search

Constraint satisfaction generates such derivation trees efficiently:

$$\dfrac{\dfrac{\dots}{u_{i_1} : \Theta_{i_1,0_\mathcal{I}}} \quad \dfrac{\dots}{v_{i_1} : \Theta_{i_1,1_\mathcal{I}}} \quad \dots \quad \dfrac{\dots}{u_{i_n} : \Theta_{i_n,0_\mathcal{I}}} \quad \dfrac{\dots}{v_{i_n} : \Theta_{i_n,1_\mathcal{I}}} \quad \dfrac{\dots}{w : \Psi}}{hcomp\ (u_{i_1}, v_{i_1}, \dots, u_{i_n}, v_{i_n})\ w : \Phi}$$

To construct an open $(n+1)$-dimensional cube, the CSP has

- $1 + 2 \cdot n$ variables
- $2 \cdot n + 4 \cdot \binom{n}{2}$ constraints
- For any $a : \Psi$ in Γ, n-th Dedekind number$^{\dim(\Psi)}$ terms

Geometric proof search

Constraint satisfaction generates such derivation trees efficiently:

$$\dfrac{\dfrac{\dots}{u_{i_1} : \Theta_{i_1,0_{\mathcal{I}}}} \quad \dfrac{\dots}{v_{i_1} : \Theta_{i_1,1_{\mathcal{I}}}} \quad \dots \quad \dfrac{\dots}{u_{i_n} : \Theta_{i_n,0_{\mathcal{I}}}} \quad \dfrac{\dots}{v_{i_n} : \Theta_{i_n,1_{\mathcal{I}}}} \quad \dfrac{\dots}{w : \Psi}}{hcomp\ (u_{i_1}, v_{i_1}, \dots, u_{i_n}, v_{i_n})\ w : \Phi}$$

To construct an open $(n+1)$-dimensional cube, the CSP has

- $1 + 2 \cdot n$ variables
- $2 \cdot n + 4 \cdot \binom{n}{2}$ constraints
- For any $a : \Psi$ in Γ, n-th Dedekind number$^{\dim(\Psi)}$ terms

For $n = 5$, 7581 terms. For $n = 6$, 7828354 terms. For $n = 7$, 2414682040998 terms. For $n = 8$, ???

Next steps

- Search S more cleverly without generating full lattice.
- Implement *transp*. Also $\sim$, *hfill* and other non-primitives?
- Construct nested Kan composition and composed cubes.
- Support different flavours of cubical type theory.
- Implement as tactic, or combine with Agsy?

The role of cubical reasoning

We have a compelling story about path induction. Can a similarly appealing story be told for cubical Kan fillings?

- [Licata and Brunerie, 2015], [Mörtberg and Pujet, 2020], ... show that cubical reasoning allows for succinct proofs.
- Heterogeneous equality is naturally expressed with cubes.
- Most natural syntax for combinatorial reasoning about spaces?

Conclusions

First step towards mechanization of higher-dimensional type theory. Desirable for several reasons:

- Lower barrier of entry to Cubical Agda.
- Understand complexity of different type theories.
- Destill syntax agnostic to concrete type theory?

Cubical Agda is highly amenable to automation!

References

Licata, D. R. and Brunerie, G. (2015).
A cubical approach to synthetic homotopy theory.
In *2015 30th Annual ACM/IEEE Symposium on Logic in Computer Science*, pages 92–103. IEEE.

Mörtberg, A. and Pujet, L. (2020).
Cubical synthetic homotopy theory.
In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, pages 158–171, New York, NY, USA. Association for Computing Machinery.

Schrijvers, T., Stuckey, P., and Wadler, P. (2009).
Monadic constraint programming.
Journal of Functional Programming, 19(6):663–697.

Agda input

```
{-# OPTIONS --cubical #-}

module Cauto.Examples.Demo where

open import Cauto.Prelude

data Triangle : Set where
  x y z : Triangle
  f : x ≡ y
  g : y ≡ z
  h : x ≡ z
  alpha : PathP (λ j → Path Triangle (f j) (h j)) (λ i → x) g

beta : PathP (λ j → Path Triangle (f j) z) h g
beta = ?
```

Agda output

```
clpc60[/home/scratch/maxore/csolver]$ ./csolver-exe ../Dropbox/Uni/experiments/Cauto/Examples/Demo.agda
λ i j → hcomp (λ k → λ {
   (i = i0) → alpha (j) (k)
 ; (i = i1) → alpha (k ∨ j) (k ∧ j)
 ; (j = i0) → f (k ∧ i)
 ; (j = i1) → g (k)
}) (f (j))
```

Verbose output I

```
clpc60[/home/scratch/maxore/csolver]$ ./csolver-exe ../Dropbox/Uni/experiments/Cauto/Examples/Demo.agda --verbose
CONTEXT
x : Point
y : Point
z : Point
f : Path Point x y
g : Path Point y z
h : Path Point x z
alpha : Path (Path Point (f<[[1]]>) (h<[[1]]>)) \1.x g
GOAL
Path (Path Point (f<[[1]]>) z) h g
SHAPES
\2.\1.x : Path (Path Point x x) \1.x \1.x
\2.\1.y : Path (Path Point y y) \1.y \1.y
\2.\1.z : Path (Path Point z z) \1.z \1.z
\2.\1.(f<[[1]]>) : Path (Path Point x y) f f
\2.\1.(f<[[2]]>) : Path (Path Point (f<[[1]]>) (f<[[1]]>)) \1.x \1.y
\2.\1.(f<[[1,2]]>) : Path (Path Point x (f<[[1]]>)) \1.x f
\2.\1.(f<[[1],[2]]>) : Path (Path Point (f<[[1]]>) y) f \1.y
\2.\1.(g<[[1]]>) : Path (Path Point y z) g g
\2.\1.(g<[[2]]>) : Path (Path Point (g<[[1]]>) (g<[[1]]>)) \1.y \1.z
\2.\1.(g<[[1,2]]>) : Path (Path Point y (g<[[1]]>)) \1.y g
\2.\1.(g<[[1],[2]]>) : Path (Path Point (g<[[1]]>) z) g \1.z
\2.\1.(h<[[1]]>) : Path (Path Point x z) h h
\2.\1.(h<[[2]]>) : Path (Path Point (h<[[1]]>) (h<[[1]]>)) \1.x \1.z
\2.\1.(h<[[1,2]]>) : Path (Path Point x (h<[[1]]>)) \1.x h
\2.\1.(h<[[1],[2]]>) : Path (Path Point (h<[[1]]>) z) h \1.z
\2.\1.((alpha<[[1]]>)<[[1]]>) : Path (Path Point x z) \1.((alpha<[[1]]>)<[[1]]>) \1.((alpha<[[1]]>)<[[1]]>)
\2.\1.((alpha<[[1]]>)<[[2]]>) : Path (Path Point x (g<[[1]]>)) f h
\2.\1.((alpha<[[1]]>)<[[1,2]]>) : Path (Path Point x (g<[[1]]>)) f \1.((alpha<[[1]]>)<[[1]]>)
\2.\1.((alpha<[[1]]>)<[[1],[2]]>) : Path (Path Point x z) \1.((alpha<[[1]]>)<[[1]]>) h
\2.\1.((alpha<[[2]]>)<[[1]]>) : Path (Path Point (f<[[1]]>) (h<[[1]]>)) \1.x g
\2.\1.((alpha<[[2]]>)<[[2]]>) : Path (Path Point ((alpha<[[1]]>)<[[1]]>) ((alpha<[[1]]>)<[[1]]>)) \1.x \1.z
\2.\1.((alpha<[[2]]>)<[[1,2]]>) : Path (Path Point (f<[[1]]>) ((alpha<[[1]]>)<[[1]]>)) \1.x g
\2.\1.((alpha<[[2]]>)<[[1],[2]]>) : Path (Path Point ((alpha<[[1]]>)<[[1]]>) (h<[[1]]>)) \1.x \1.z
\2.\1.((alpha<[[1,2]]>)<[[1]]>) : Path (Path Point x (h<[[1]]>)) \1.x \1.((alpha<[[1]]>)<[[1]]>)
\2.\1.((alpha<[[1,2]]>)<[[2]]>) : Path (Path Point x ((alpha<[[1]]>)<[[1]]>)) \1.x h
\2.\1.((alpha<[[1,2]]>)<[[1,2]]>) : Path (Path Point x ((alpha<[[1]]>)<[[1]]>)) \1.x \1.((alpha<[[1]]>)<[[1]]>)
\2.\1.((alpha<[[1,2]]>)<[[1],[2]]>) : Path (Path Point x (h<[[1]]>)) \1.x h
\2.\1.((alpha<[[1],[2]]>)<[[1]]>) : Path (Path Point (f<[[1]]>) z) \1.((alpha<[[1]]>)<[[1]]>) g
\2.\1.((alpha<[[1],[2]]>)<[[2]]>) : Path (Path Point ((alpha<[[1]]>)<[[1]]>) (g<[[1]]>)) f \1.z
\2.\1.((alpha<[[1],[2]]>)<[[1,2]]>) : Path (Path Point (f<[[1]]>) (g<[[1]]>)) f g
\2.\1.((alpha<[[1],[2]]>)<[[1],[2]]>) : Path (Path Point ((alpha<[[1]]>)<[[1]]>) z) \1.((alpha<[[1]]>)<[[1]]>) \1.z
NO DIRECT FIT FOUND, SEARCHING FOR HIGHER CUBES
```

Verbose output II

```
Path (Path Point (f<[[1]]>) z) h g
DOMAINS BEFORE CONSTRAINTS
i0: fromList [\2.\1.(h<[[2]]>),\2.\1.(h<[[1,2]]>),\2.\1.((alpha<[[2]]>)<[[1]]>),\2.\1.((alpha<[[2]]>)<[[1],[2]]>),\2
.\1.((alpha<[[1,2]]>)<[[1]]>),\2.\1.((alpha<[[1,2]]>)<[[1],[2]]>)]
j0: fromList [\2.\1.(f<[[2]]>),\2.\1.(f<[[1,2]]>)]
i1: fromList [\2.\1.(g<[[2]]>),\2.\1.(g<[[1,2]]>),\2.\1.((alpha<[[1]]>)<[[2]]>),\2.\1.((alpha<[[1]]>)<[[1,2]]>),\2.\
1.((alpha<[[1],[2]]>)<[[2]]>),\2.\1.((alpha<[[1],[2]]>)<[[1,2]]>)]
j1: fromList [\2.\1.z,\2.\1.(g<[[1]]>),\2.\1.(g<[[1],[2]]>),\2.\1.(h<[[1]]>),\2.\1.(h<[[1],[2]]>),\2.\1.((alpha<[[1]
]>)<[[1]]>),\2.\1.((alpha<[[1]]>)<[[1],[2]]>),\2.\1.((alpha<[[1],[2]]>)<[[1]]>),\2.\1.((alpha<[[1],[2]]>)<[[1],[2]]>
)]
DOMAINS AFTER BACK CONSTRAINTS
fromList [\2.\1.x,\2.\1.(f<[[1]]>),\2.\1.(f<[[2]]>),\2.\1.(f<[[1,2]]>),\2.\1.(f<[[1],[2]]>),\2.\1.((alpha<[[1]]>)<[[
1]]>),\2.\1.((alpha<[[1]]>)<[[1,2]]>),\2.\1.((alpha<[[2]]>)<[[1]]>),\2.\1.((alpha<[[2]]>)<[[1,2]]>),\2.\1.((alpha<[[
1,2]]>)<[[1]]>),\2.\1.((alpha<[[1,2]]>)<[[1,2]]>),\2.\1.((alpha<[[1],[2]]>)<[[1]]>),\2.\1.((alpha<[[1],[2]]>)<[[1,2]
]>)]
i0: fromList [\2.\1.(h<[[2]]>),\2.\1.(h<[[1,2]]>),\2.\1.((alpha<[[2]]>)<[[1]]>),\2.\1.((alpha<[[2]]>)<[[1],[2]]>),\2
.\1.((alpha<[[1,2]]>)<[[1]]>),\2.\1.((alpha<[[1,2]]>)<[[1],[2]]>)]
j0: fromList [\2.\1.(f<[[2]]>),\2.\1.(f<[[1,2]]>)]
i1: fromList [\2.\1.(g<[[2]]>),\2.\1.(g<[[1,2]]>),\2.\1.((alpha<[[1]]>)<[[2]]>),\2.\1.((alpha<[[1]]>)<[[1,2]]>),\2.\
1.((alpha<[[1],[2]]>)<[[2]]>),\2.\1.((alpha<[[1],[2]]>)<[[1,2]]>)]
j1: fromList [\2.\1.z,\2.\1.(g<[[1]]>),\2.\1.(g<[[1],[2]]>),\2.\1.(h<[[1]]>),\2.\1.(h<[[1],[2]]>),\2.\1.((alpha<[[1]
]>)<[[1]]>),\2.\1.((alpha<[[1]]>)<[[1],[2]]>),\2.\1.((alpha<[[1],[2]]>)<[[1]]>),\2.\1.((alpha<[[1],[2]]>)<[[1],[2]]>
)]
DOMAINS AFTER SIDE CONSTRAINTS
fromList [\2.\1.(f<[[1]]>),\2.\1.(f<[[1,2]]>)]
i0: fromList [\2.\1.((alpha<[[2]]>)<[[1]]>),\2.\1.((alpha<[[1,2]]>)<[[1]]>)]
j0: fromList [\2.\1.(f<[[1,2]]>)]
i1: fromList [\2.\1.((alpha<[[1],[2]]>)<[[1,2]]>)]
j1: fromList [\2.\1.(g<[[1]]>),\2.\1.((alpha<[[1],[2]]>)<[[1]]>)]
[Comp \2.\1.(f<[[1]]>) [(\2.\1.((alpha<[[2]]>)<[[1]]>),\2.\1.((alpha<[[1],[2]]>)<[[1,2]]>)),(\2.\1.(f<[[1,2]]>),\2.\
1.(g<[[1]]>))]]
λ i j → hcomp (λ k → λ {
   (i = i0) → alpha (j) (k)
 ; (i = i1) → alpha (k ∨ j) (k ∧ j)
 ; (j = i0) → f (k ∧ i)
 ; (j = i1) → g (k)
}) (f (j))
```

Evaluation

$$r[i = 0_{\mathcal{I}}] \hookrightarrow \begin{cases} 0_{\mathcal{I}} \text{ if } i \in c \text{ for all } c \in r \\ \{c \mid i \notin c\} \text{ otherwise} \end{cases}$$

$$r[i = 1_{\mathcal{I}}] \hookrightarrow \begin{cases} 1_{\mathcal{I}} \text{ if } c = \{i\} \text{ for some } c \in r \\ \{c' \mid c' = c \backslash \{i\} \text{ for } c \in r\} \text{ otherwise} \end{cases}$$

$$a[i = e] \hookrightarrow a$$

$$(\lambda j.t)[i = e] \hookrightarrow \lambda j.t[i = e] \qquad (i \neq$$

$$(t\ r)[i = e] \hookrightarrow \begin{cases} u \text{ if } r[i = e] \hookrightarrow 0_{\mathcal{I}} \text{ and } t : \text{PathP } (\lambda k.\Phi)\ u\ v \\ v \text{ if } r[i = e] \hookrightarrow 1_{\mathcal{I}} \text{ and } t : \text{PathP } (\lambda k.\Phi)\ u\ v \\ (t[i = e])\ (r[i = e]) \text{ otherwise} \end{cases} \qquad (i \neq$$