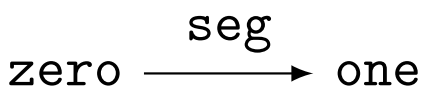
Computational semantics of Cubical Inductive Types

Evan Cavallo
with Bob Harper

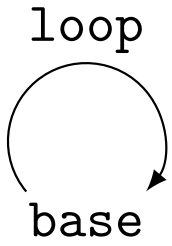
Carnegie Mellon University

Mar 23, 2018 @ MURI

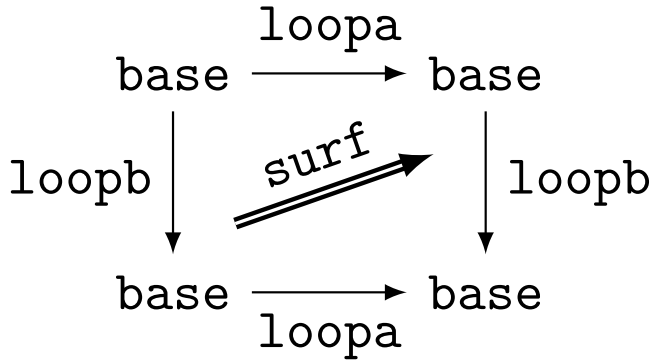
Higher Inductive Types



(a) Interval



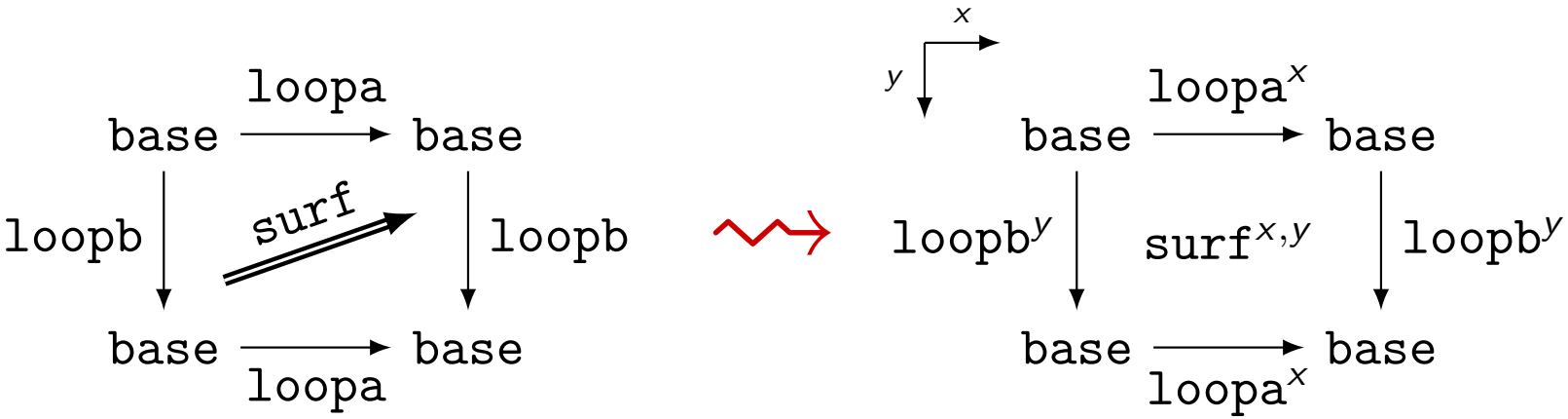
(b) Circle



(c) Torus

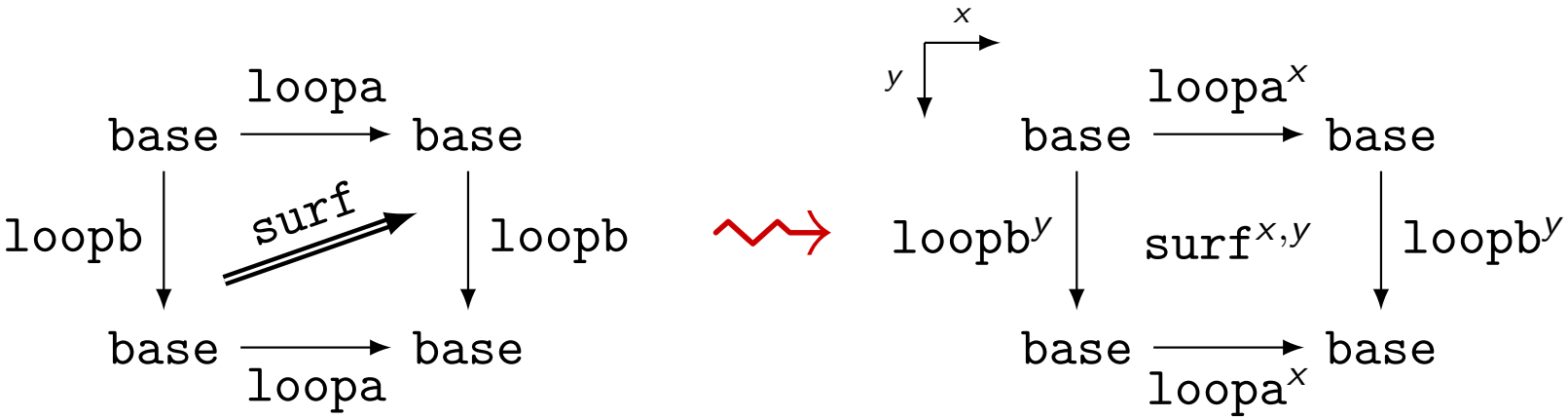
- Types generated by specified points and paths
  - Homotopy-initial algebras [AGS17]
  - Weakly stable typal initial algebras [LS17]
- Can we regard these as computational objects?

Cubical Inductive Types



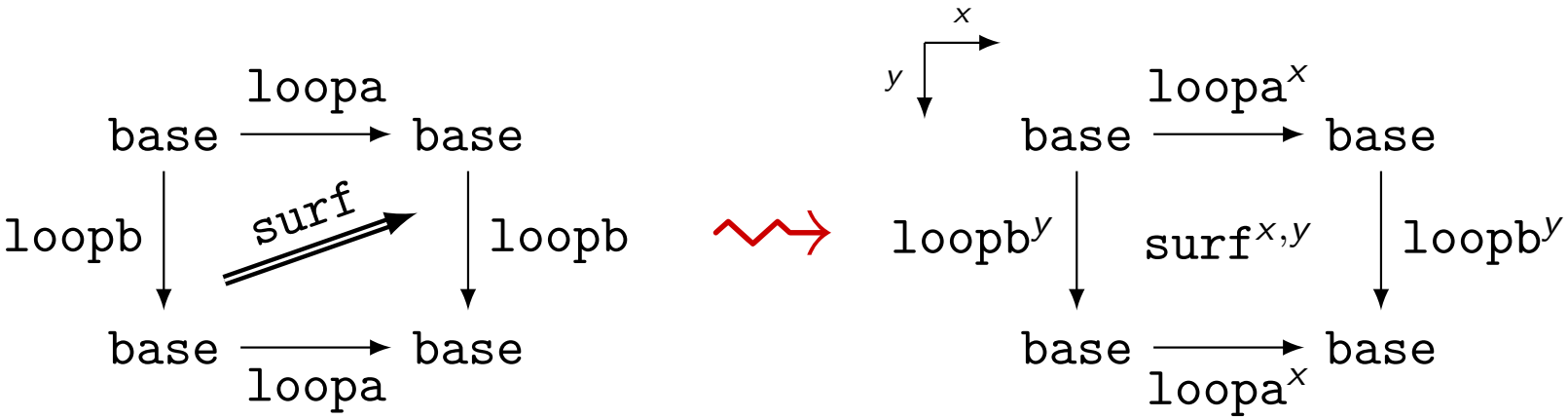
1. Choose a format for organizing higher structure

Cubical Inductive Types



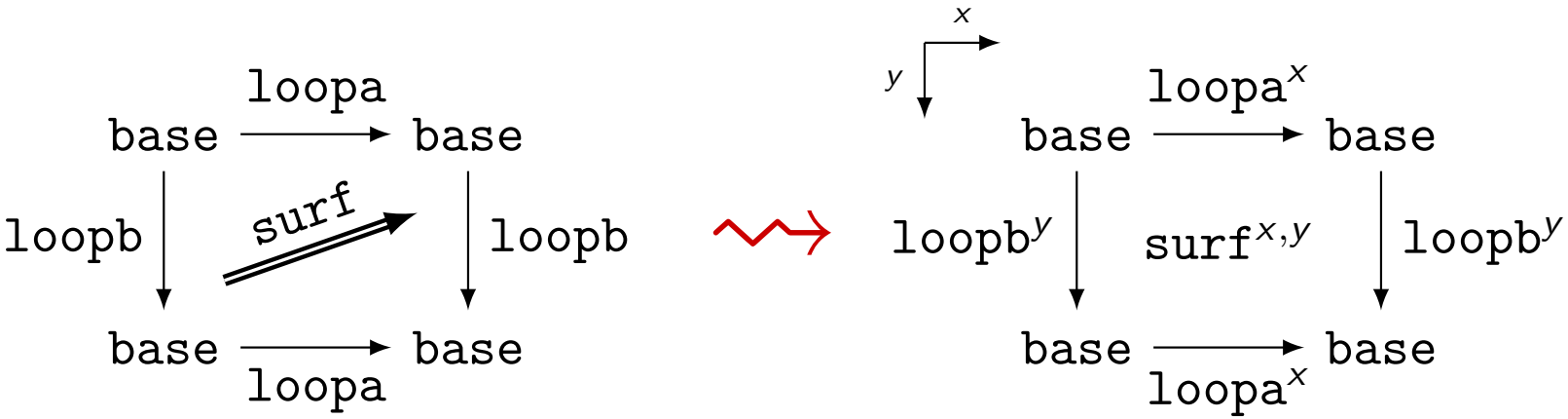
1. Choose a format for organizing higher structure
2. Define a language for specifying cell constructors in this format

Cubical Inductive Types



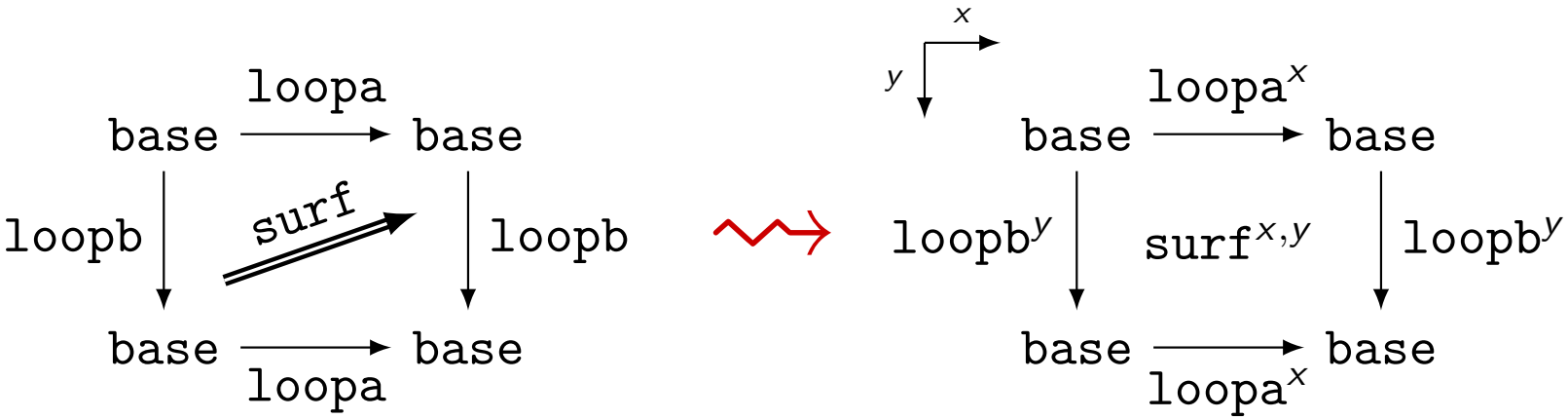
1. Choose a format for organizing higher structure
2. Define a language for specifying cell constructors in this format
3. Define (in a computational semantics) the inductive type generated by each specification

Cubical Inductive Types



1. Choose a format for organizing higher structure
2. Define a language for specifying cell constructors in this format
3. Define (in a computational semantics) the inductive type generated by each specification
4. Use these as higher inductive types (or don't)

Cubical Inductive Types



1. Choose a format for organizing higher structure
2. Define a language for specifying cell constructors in this format
3. Define (in a computational semantics) the inductive type generated by each specification
4. Use these as higher inductive types (or don't)

Feature Requests for Cubical Inductive Types

Feature Requests for Cubical Inductive Types

- Higher-dimensional constructors with boundaries...

$$\frac{}{\texttt{base} \xrightarrow{\texttt{loop}^x} \texttt{base} \;\in \mathbb{S}^1}$$

Feature Requests for Cubical Inductive Types

- Higher-dimensional constructors with boundaries...

$$\frac{}{\texttt{base} \xrightarrow{\texttt{loop}^x} \texttt{base} \;\; \in \mathbb{S}^1}$$

- ...which can use arbitrary functions...

$$\frac{P \in C}{\texttt{left}(FP) \xrightarrow{\texttt{glue}^x(P)} \texttt{right}(GP) \;\; \in A \sqcup_C^{F,G} B}$$

Feature Requests for Cubical Inductive Types

- Higher-dimensional constructors with boundaries...

$$\frac{}{\texttt{base} \xrightarrow{\texttt{loop}^x} \texttt{base} \;\; \in \mathbb{S}^1}$$

- ...which can use arbitrary functions...

$$\frac{P \in C}{\texttt{left}(FP) \xrightarrow{\texttt{glue}^x(P)} \texttt{right}(GP) \;\; \in A \sqcup_C^{F,G} B}$$

- ...or recursive arguments.

$$\frac{M \in \|A\| \qquad N \in \|A\|}{M \xrightarrow{\texttt{path}^x(M;N)} N \;\; \in \|A\|}$$

Schemata for Inductive Types

Schemata for Inductive Types

- W-types [ML82]:

$$\frac{A\ \mathtt{type} \qquad a : A \gg B\ \mathtt{type}}{\mathtt{W}(A; a.B)\ \mathtt{type}}$$

$$\frac{M \in A \qquad F \in B[M/a] \to \mathtt{W}(A; a.B)}{\mathtt{sup}(M; F) \in \mathtt{W}(A; a.B)}$$

Schemata for Inductive Types

- W-types [ML82]:

$$\frac{A\ \mathtt{type} \qquad a : A \gg B\ \mathtt{type}}{\mathtt{W}(A; a.B)\ \mathtt{type}}$$

$$\frac{M \in A \qquad F \in B[M/a] \to \mathtt{W}(A; a.B)}{\mathtt{sup}(M; F) \in \mathtt{W}(A; a.B)}$$

$$\begin{array}{lcl} \mathtt{nat} & := & \mathtt{W}(\mathtt{bool}; b.\mathtt{if}(b; \mathtt{void}, \mathtt{unit})) \\ \mathtt{zero} & := & \mathtt{sup}(\mathtt{true}; \lambda e.\mathtt{void\text{-}elim}(e)) \\ \mathtt{s}(M) & := & \mathtt{sup}(\mathtt{false}; \lambda\_.M) \end{array}$$

Schemata for Inductive Types

- More explicit schemata [CP88; Dyb94]:

$$\begin{array}{l}\texttt{data X where}\\ \quad \texttt{intro}_1 \in (a_1 : A_1) \cdots (a_n : A_n) \to (B_1 \to \texttt{X}) \to \cdots \to (B_k \to \texttt{X}) \to \texttt{X}\\ \quad \vdots\\ \quad \texttt{intro}_m \in \cdots\end{array}$$

Schemata for Inductive Types

- More explicit schemata [CP88; Dyb94]:

$$\begin{array}{l}\texttt{data X where}\\ \quad \texttt{intro}_1 \in (a_1 : A_1) \cdots (a_n : A_n)(f_1 : \mathrm{B}_1) \cdots (f_k : \mathrm{B}_k) \to \texttt{X}\\ \quad \vdots\\ \quad \texttt{intro}_\texttt{m} \in \cdots\end{array}$$

where

$$\mathrm{C} \quad ::= \quad \texttt{X} \mid (b : B) \to \mathrm{C}$$

Schemata for Inductive Types

- More explicit schemata [CP88; Dyb94]:

$$\begin{array}{l}\texttt{data X where}\\ \quad \texttt{intro}_1 \in (a_1 : A_1) \cdots (a_n : A_n)(f_1 : \mathrm{B}_1) \cdots (f_k : \mathrm{B}_k) \to \texttt{X}\\ \quad \vdots\\ \quad \texttt{intro}_\texttt{m} \in \cdots\end{array}$$

where

$$\mathrm{C} \quad ::= \quad \texttt{X} \mid (b : B) \to \mathrm{C} \quad (+ \text{ typing rules for } \mathrm{C} \text{ atype})$$

Adding Dimensions and Boundaries

$$\begin{array}{l}\texttt{data X where}\\ \quad \texttt{intro}_1 \in (a_1 : A_1) \cdots (a_n : A_n)(f_1 : \mathrm{C}_1) \cdots (f_k : \mathrm{C}_k) \to \texttt{X}\\ \quad \vdots\\ \quad \texttt{intro}_\texttt{m} \in \cdots\end{array}$$

$$\mathrm{C} \quad ::= \quad \texttt{X} \mid (b : B) \to \mathrm{C} \quad (+ \text{ typing rules for } \mathrm{C} \text{ atype})$$

Adding Dimensions and Boundaries

$$
\begin{array}{l}
\texttt{data X where} \\
\quad \texttt{intro}_1{}^{x_1,\ldots,x_l} \in (a_1 : A_1) \cdots (a_n : A_n)(f_1 : \mathrm{C}_1) \cdots (f_k : \mathrm{C}_k) \to \texttt{X} \\
\quad \vdots \\
\quad \texttt{intro}_m{}^{\cdots} \in \cdots
\end{array}
$$

$$\mathrm{C} \quad ::= \quad \texttt{X} \mid (b : B) \to \mathrm{C} \quad (+ \text{ typing rules for } \mathrm{C} \text{ atype})$$

Adding Dimensions and Boundaries

$$
\begin{array}{l}
\texttt{data X where} \\
\quad \texttt{intro}_1{}^{x_1,\ldots,x_l} \in (a_1 : A_1) \cdots (a_n : A_n)(f_1 : \mathrm{C}_1) \cdots (f_k : \mathrm{C}_k) \to \texttt{X} \\
\qquad [\xi_1 \hookrightarrow \mathrm{M}_1, \ldots, \xi_p \hookrightarrow \mathrm{M}_p] \\
\quad \vdots \\
\quad \texttt{intro}_{\mathtt{m}}{}^{\cdots} \in \cdots \\
\qquad [\cdots]
\end{array}
$$

$$
\begin{array}{lcl}
\mathrm{C} & ::= & \texttt{X} \mid (b : B) \to \mathrm{C} \quad (+ \text{ typing rules for } \mathrm{C} \text{ atype}) \\
\xi & ::= & r = r' \\
\mathrm{M} & ::= & \texttt{intro}_i(\cdots) \mid \texttt{fcom}(\cdots) \mid \lambda a.\mathrm{M} \mid \texttt{app}(\mathrm{M}, M) \\
 & & (+ \text{ typing rules for } \mathrm{M} : \mathrm{C})
\end{array}
$$

Formulating elimination

$$\texttt{data}\ A \sqcup_C^{F,G} B\ \texttt{where}$$

$$\texttt{left} \in (a : A) \to A \sqcup_C^{F,G} B$$

$$\texttt{right} \in (b : B) \to A \sqcup_C^{F,G} B$$

$$\texttt{glue}^x \in (c : C) \to A \sqcup_C^{F,G} B$$

$$[x = 0 \hookrightarrow \texttt{left}(Fc), x = 1 \hookrightarrow \texttt{right}(Gc)]$$

Formulating elimination

$$\begin{array}{l}\texttt{data}\ A \sqcup_C^{F,G} B\ \texttt{where} \\ \quad \texttt{left} \in (a : A) \to A \sqcup_C^{F,G} B \\ \quad \texttt{right} \in (b : B) \to A \sqcup_C^{F,G} B \\ \quad \texttt{glue}^x \in (c : C) \to A \sqcup_C^{F,G} B \\ \qquad [x = 0 \hookrightarrow \texttt{left}(Fc), x = 1 \hookrightarrow \texttt{right}(Gc)]\end{array}$$

To define a function $(u : A \sqcup_C^{F,G} B) \to P(u)$, provide:

Formulating elimination

$$\begin{aligned}
&\texttt{data } A \sqcup_C^{F,G} B \texttt{ where} \\
&\quad \texttt{left} \in (a : A) \to A \sqcup_C^{F,G} B \\
&\quad \texttt{right} \in (b : B) \to A \sqcup_C^{F,G} B \\
&\quad \texttt{glue}^x \in (c : C) \to A \sqcup_C^{F,G} B \\
&\qquad [x = 0 \hookrightarrow \texttt{left}(Fc), x = 1 \hookrightarrow \texttt{right}(Gc)]
\end{aligned}$$

To define a function $(u : A \sqcup_C^{F,G} B) \to P(u)$, provide:

- $l : (a : A) \to P(\texttt{left}(a))$,

Formulating elimination

$$\begin{aligned}
&\texttt{data } A \sqcup_C^{F,G} B \texttt{ where} \\
&\quad \texttt{left} \in (a : A) \to A \sqcup_C^{F,G} B \\
&\quad \texttt{right} \in (b : B) \to A \sqcup_C^{F,G} B \\
&\quad \texttt{glue}^x \in (c : C) \to A \sqcup_C^{F,G} B \\
&\qquad [x = 0 \hookrightarrow \texttt{left}(Fc), x = 1 \hookrightarrow \texttt{right}(Gc)]
\end{aligned}$$

To define a function $(u : A \sqcup_C^{F,G} B) \to P(u)$, provide:

- $l : (a : A) \to P(\texttt{left}(a))$,
- $r : (b : B) \to P(\texttt{right}(b))$,

Formulating elimination

$$
\begin{aligned}
&\texttt{data } A \sqcup_C^{F,G} B \texttt{ where} \\
&\quad \texttt{left} \in (a : A) \to A \sqcup_C^{F,G} B \\
&\quad \texttt{right} \in (b : B) \to A \sqcup_C^{F,G} B \\
&\quad \texttt{glue}^x \in (c : C) \to A \sqcup_C^{F,G} B \\
&\qquad [x = 0 \hookrightarrow \texttt{left}(Fc), x = 1 \hookrightarrow \texttt{right}(Gc)]
\end{aligned}
$$

To define a function $(u : A \sqcup_C^{F,G} B) \to P(u)$, provide:

- $l : (a : A) \to P(\texttt{left}(a))$,
- $r : (b : B) \to P(\texttt{right}(b))$,
- $g^x : (c : C) \to P(\texttt{glue}^x(c))$, such that

Formulating elimination

$$\begin{aligned}
&\texttt{data } A \sqcup_C^{F,G} B \texttt{ where} \\
&\quad \texttt{left} \in (a : A) \to A \sqcup_C^{F,G} B \\
&\quad \texttt{right} \in (b : B) \to A \sqcup_C^{F,G} B \\
&\quad \texttt{glue}^x \in (c : C) \to A \sqcup_C^{F,G} B \\
&\qquad [x = 0 \hookrightarrow \texttt{left}(Fc), x = 1 \hookrightarrow \texttt{right}(Gc)]
\end{aligned}$$

To define a function $(u : A \sqcup_C^{F,G} B) \to P(u)$, provide:

- $l : (a : A) \to P(\texttt{left}(a))$,
- $r : (b : B) \to P(\texttt{right}(b))$,
- $g^x : (c : C) \to P(\texttt{glue}^x(c))$, such that
  - $g^0(c) \doteq l(Fc) \in P(\texttt{glue}^0(c))$,

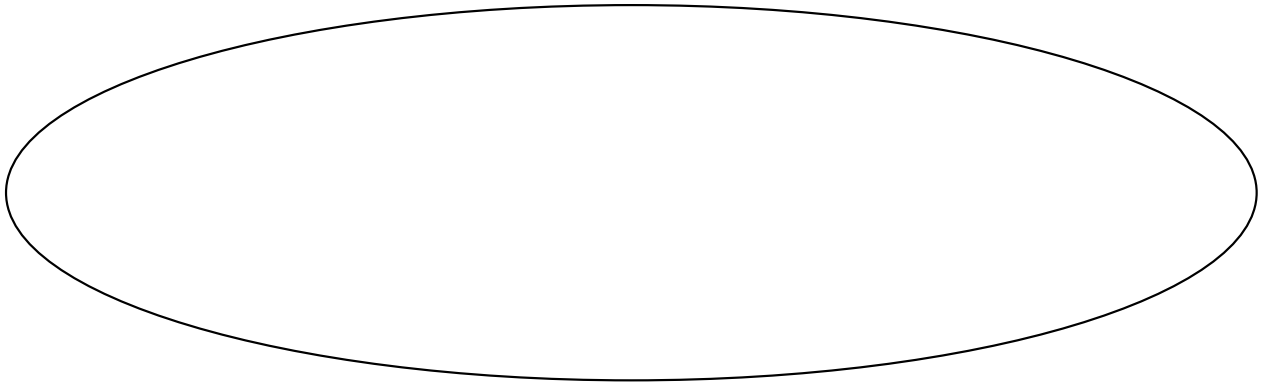
Formulating elimination

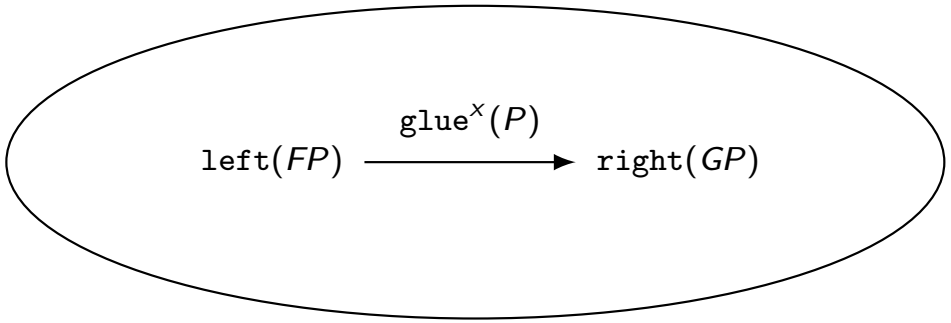
$$\begin{aligned}
&\texttt{data } A \sqcup_C^{F,G} B \texttt{ where} \\
&\quad \texttt{left} \in (a : A) \to A \sqcup_C^{F,G} B \\
&\quad \texttt{right} \in (b : B) \to A \sqcup_C^{F,G} B \\
&\quad \texttt{glue}^x \in (c : C) \to A \sqcup_C^{F,G} B \\
&\qquad [x = 0 \hookrightarrow \texttt{left}(Fc), x = 1 \hookrightarrow \texttt{right}(Gc)]
\end{aligned}$$

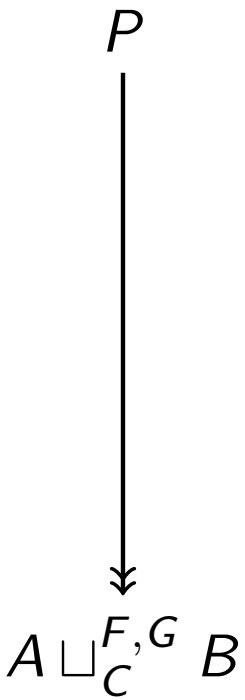
To define a function $(u : A \sqcup_C^{F,G} B) \to P(u)$, provide:

- $l : (a : A) \to P(\texttt{left}(a))$,
- $r : (b : B) \to P(\texttt{right}(b))$,
- $g^x : (c : C) \to P(\texttt{glue}^x(c))$, such that
  - $g^0(c) \doteq l(Fc) \in P(\texttt{glue}^0(c))$,
  - $g^1(c) \doteq r(Gc) \in P(\texttt{glue}^1(c))$.

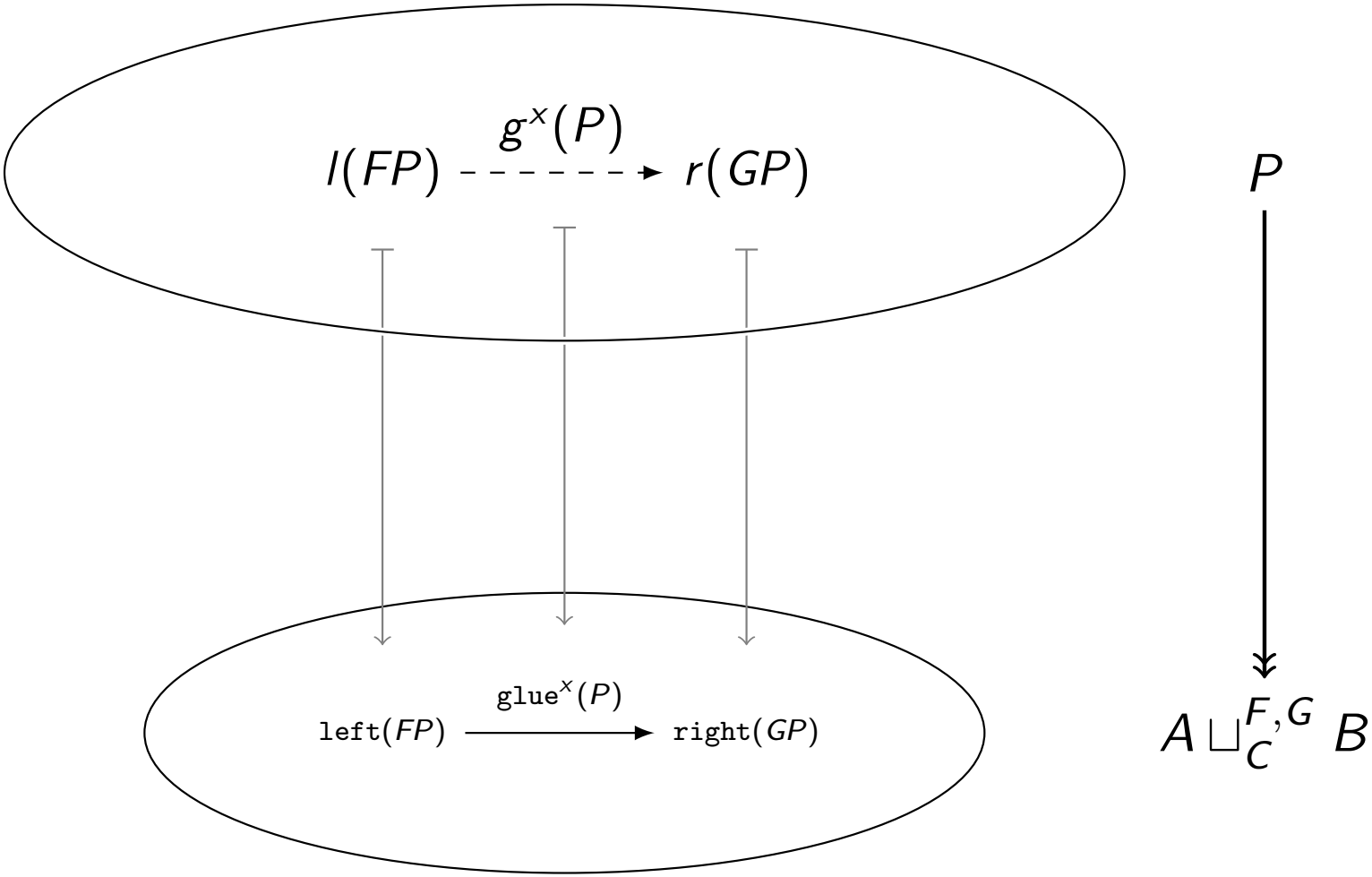
Formulating elimination







Formulating elimination



Formulating elimination

$$
\begin{array}{l}
\texttt{data}\ \|A\|\ \texttt{where} \\
\quad \texttt{pt} \in (a : A) \to \|A\| \\
\quad \texttt{path}^x \in (b : \|A\|)(c : \|A\|) \to \|A\| \\
\qquad [x = 0 \hookrightarrow b, x = 1 \hookrightarrow c]
\end{array}
$$

Formulating elimination

$$
\begin{aligned}
&\texttt{data}\ \|A\|\ \texttt{where}\\
&\quad \texttt{pt} \in (a : A) \to \|A\|\\
&\quad \texttt{path}^x \in (b : \|A\|)(c : \|A\|) \to \|A\|\\
&\qquad [x = 0 \hookrightarrow b, x = 1 \hookrightarrow c]
\end{aligned}
$$

To define a function $(u : \|A\|) \to P(u)$, provide:

Formulating elimination

$$
\begin{aligned}
&\texttt{data } \|A\| \texttt{ where} \\
&\quad \texttt{pt} \in (a : A) \to \|A\| \\
&\quad \texttt{path}^x \in (b : \|A\|)(c : \|A\|) \to \|A\| \\
&\qquad [x = 0 \hookrightarrow b, x = 1 \hookrightarrow c]
\end{aligned}
$$

To define a function $(u : \|A\|) \to P(u)$, provide:

- $p : (a : A) \to P(\texttt{pt}(a))$,

Formulating elimination

$$
\begin{aligned}
&\texttt{data } \|A\| \texttt{ where} \\
&\quad \texttt{pt} \in (a : A) \to \|A\| \\
&\quad \texttt{path}^x \in (b : \|A\|)(c : \|A\|) \to \|A\| \\
&\qquad [x = 0 \hookrightarrow b, x = 1 \hookrightarrow c]
\end{aligned}
$$

To define a function $(u : \|A\|) \to P(u)$, provide:

- $p : (a : A) \to P(\texttt{pt}(a))$,
- $q^x : (b, c : \|A\|)(b^* : P(b))(c^* : P(c)) \to P(\texttt{path}^x(b; c))$, such that

Formulating elimination

$$
\begin{aligned}
&\texttt{data } \|A\| \texttt{ where} \\
&\quad \texttt{pt} \in (a : A) \to \|A\| \\
&\quad \texttt{path}^x \in (b : \|A\|)(c : \|A\|) \to \|A\| \\
&\qquad [x = 0 \hookrightarrow b, x = 1 \hookrightarrow c]
\end{aligned}
$$

To define a function $(u : \|A\|) \to P(u)$, provide:

- $p : (a : A) \to P(\texttt{pt}(a))$,
- $q^x : (b, c : \|A\|)(b^* : P(b))(c^* : P(c)) \to P(\texttt{path}^x(b; c))$, such that
  - $q^0(b, c, b^*, c^*) \doteq b^* \in P(\texttt{path}^0(b; c))$,

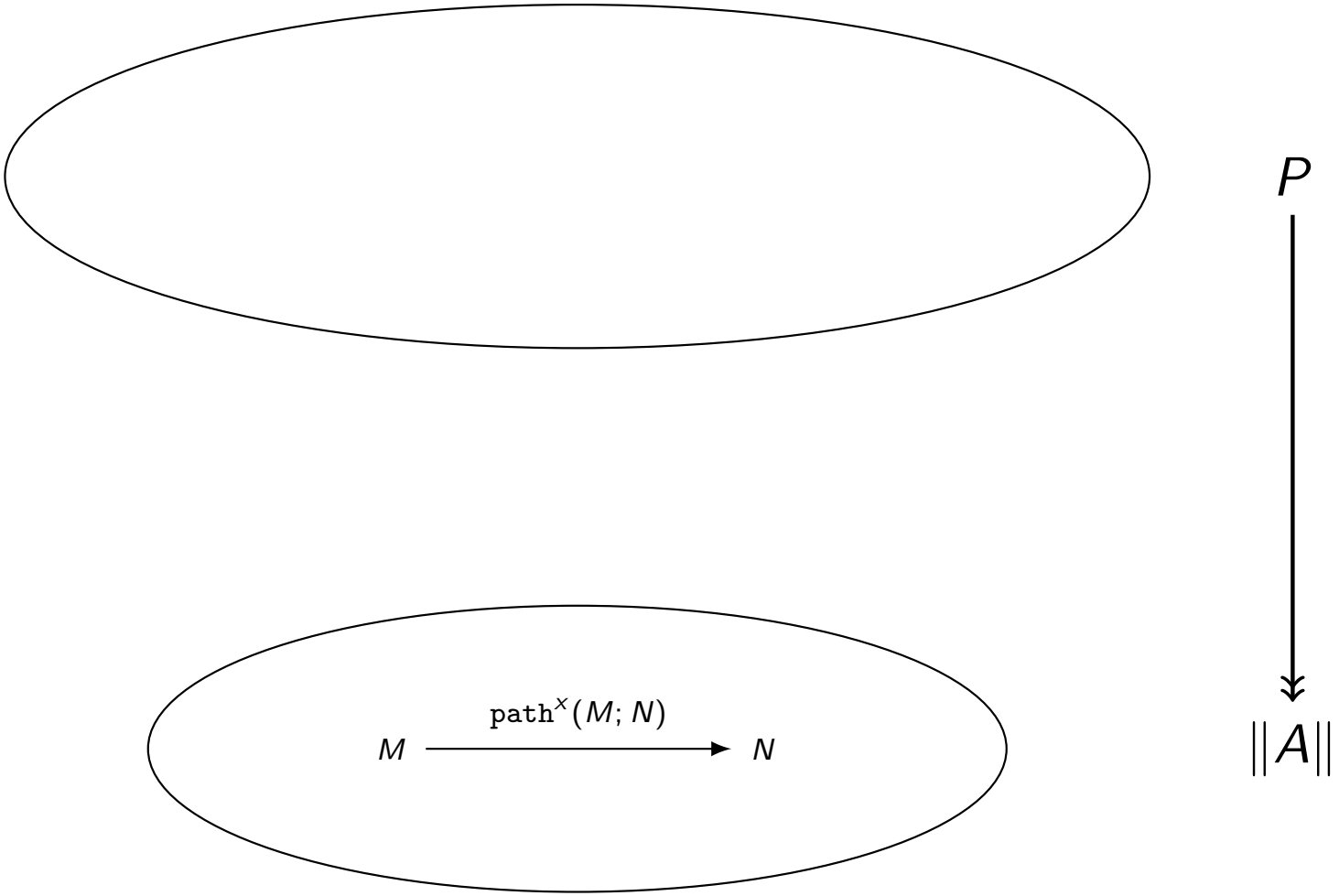
Formulating elimination

$$
\begin{aligned}
&\texttt{data } \|A\| \texttt{ where} \\
&\quad \texttt{pt} \in (a : A) \to \|A\| \\
&\quad \texttt{path}^x \in (b : \|A\|)(c : \|A\|) \to \|A\| \\
&\qquad [x = 0 \hookrightarrow b, x = 1 \hookrightarrow c]
\end{aligned}
$$

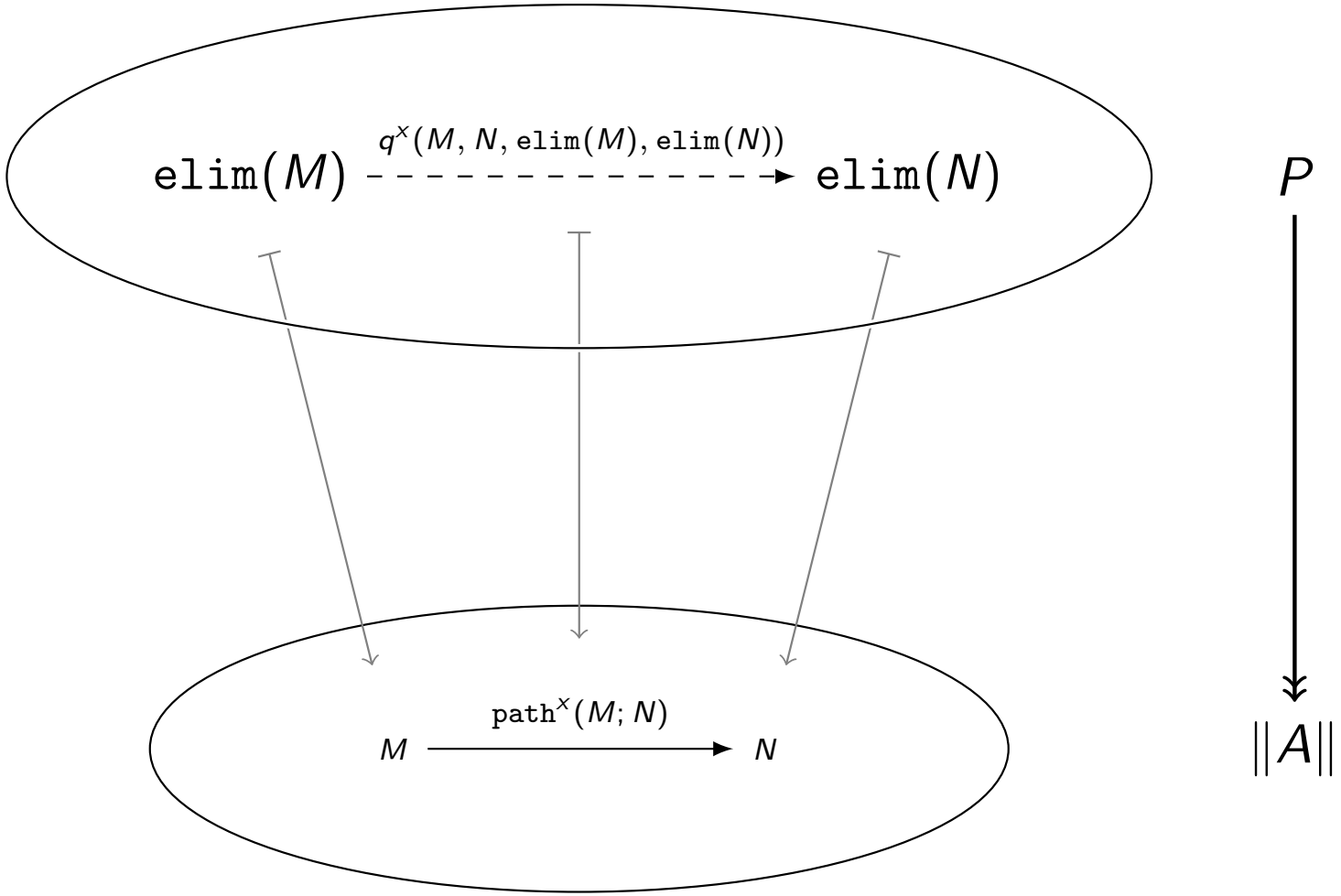
To define a function $(u : \|A\|) \to P(u)$, provide:

- $p : (a : A) \to P(\texttt{pt}(a))$,
- $q^x : (b, c : \|A\|)(b^* : P(b))(c^* : P(c)) \to P(\texttt{path}^x(b; c))$, such that
  - $q^0(b, c, b^*, c^*) \doteq b^* \in P(\texttt{path}^0(b; c))$,
  - $q^1(b, c, b^*, c^*) \doteq c^* \in P(\texttt{path}^1(b; c))$.

Formulating elimination



Formulating elimination



Formulating elimination

$$
\begin{array}{l}
\texttt{data X where} \\
\quad \texttt{intro}_1{}^{x_1,\ldots,x_l} \in (a_1 : A_1) \cdots (a_n : A_n)(f_1 : \mathrm{C}_1) \cdots (f_k : \mathrm{C}_k) \to \texttt{X} \\
\qquad [\xi_1 \hookrightarrow \mathrm{M}_1, \ldots, \xi_p \hookrightarrow \mathrm{M}_p] \\
\quad \vdots \\
\quad \texttt{intro}_{\mathtt{m}}{}^{\cdots} \in \cdots \\
\qquad [\cdots]
\end{array}
$$

To define a function $(u : \texttt{X}) \to P(u)$, provide:

Formulating elimination

$$
\begin{array}{l}
\texttt{data X where} \\
\quad \texttt{intro}_1{}^{x_1,\ldots,x_l} \in (a_1 : A_1) \cdots (a_n : A_n)(f_1 : \mathrm{C}_1) \cdots (f_k : \mathrm{C}_k) \to \texttt{X} \\
\qquad [\xi_1 \hookrightarrow \mathrm{M}_1, \ldots, \xi_p \hookrightarrow \mathrm{M}_p] \\
\quad \vdots \\
\quad \texttt{intro}_{\mathtt{m}}{}^{\cdots} \in \cdots \\
\qquad [\cdots]
\end{array}
$$

To define a function $(u : \texttt{X}) \to P(u)$, provide:

- $r_1^{x_1,\ldots,x_l} : (a_1, \ldots, a_n)(f_1, \ldots, f_k)(f_1^*, \ldots, f_k^*) \to P(\texttt{intro}_1(\ldots))$, such that

Formulating elimination

$$
\begin{array}{l}
\texttt{data X where} \\
\quad \texttt{intro}_1{}^{x_1,\ldots,x_l} \in (a_1 : A_1) \cdots (a_n : A_n)(f_1 : \mathrm{C}_1) \cdots (f_k : \mathrm{C}_k) \to \texttt{X} \\
\qquad [\xi_1 \hookrightarrow \mathrm{M}_1, \ldots, \xi_p \hookrightarrow \mathrm{M}_p] \\
\quad \vdots \\
\quad \texttt{intro}_{\mathtt{m}}{}^{\cdots} \in \cdots \\
\qquad [\cdots]
\end{array}
$$

To define a function $(u : \texttt{X}) \to P(u)$, provide:

- $r_1^{x_1,\ldots,x_l} : (a_1, \ldots, a_n)(f_1, \ldots, f_k)(f_1^*, \ldots, f_k^*) \to P(\texttt{intro}_1(\ldots))$, such that
  - $r_1^{\cdots}(\cdots) \doteq \mathrm{M}_i{}^*(\cdots) \in P(\texttt{intro}_1(\ldots))$ within ξ_i, for each i

Formulating elimination

$$\begin{array}{l}\texttt{data X where}\\ \quad \texttt{intro}_1{}^{x_1,\ldots,x_l} \in (a_1 : A_1)\cdots(a_n : A_n)(f_1 : \mathrm{C}_1)\cdots(f_k : \mathrm{C}_k) \to \texttt{X}\\ \qquad [\xi_1 \hookrightarrow \mathrm{M}_1, \ldots, \xi_p \hookrightarrow \mathrm{M}_p]\\ \quad \vdots\\ \quad \texttt{intro}_{\mathtt{m}}{}^{\cdots} \in \cdots\\ \qquad [\cdots]\end{array}$$

To define a function $(u : \texttt{X}) \to P(u)$, provide:

- $r_1^{x_1,\ldots,x_l} : (a_1, \ldots, a_n)(f_1, \ldots, f_k)(f_1^*, \ldots, f_k^*) \to P(\texttt{intro}_1(\ldots))$, such that
  - $r_1^{\cdots}(\cdots) \doteq \mathrm{M}_i{}^*(\cdots) \in P(\texttt{intro}_1(\ldots))$ within ξ_i, for each i

$\vdots$

Formulating elimination

$$
\begin{array}{l}
\texttt{data X where} \\
\quad \texttt{intro}_1{}^{x_1,\ldots,x_l} \in (a_1 : A_1) \cdots (a_n : A_n)(f_1 : \mathrm{C}_1) \cdots (f_k : \mathrm{C}_k) \to \texttt{X} \\
\qquad [\xi_1 \hookrightarrow \mathrm{M}_1, \ldots, \xi_p \hookrightarrow \mathrm{M}_p] \\
\quad \vdots \\
\quad \texttt{intro}_{\texttt{m}}{}^{\cdots} \in \cdots \\
\qquad [\cdots]
\end{array}
$$

To define a function $(u : \texttt{X}) \to P(u)$, provide:

- $r_1^{x_1,\ldots,x_l} : (a_1, \ldots, a_n)(f_1, \ldots, f_k)(f_1^*, \ldots, f_k^*) \to P(\texttt{intro}_1(\ldots))$, such that
  - $r_1^{\cdots}(\cdots) \doteq \mathrm{M}_i{}^*(\cdots) \in P(\texttt{intro}_1(\ldots))$ within ξ_i, for each i

$\vdots$

- $r_m^{\cdots} \cdots$

Examples

- Circle
- Pushouts
- (-1)-Truncations (and n-truncations, via hub-and-spokes)
- Localizations
- ~~Lumsdaine-Shulman §9~~

Computational interpretation

Computational interpretation

Given a specification $\mathcal{K} = \{\texttt{data X where } \cdots\}$, construct a type $\texttt{ind}(\mathcal{K})$ which

Computational interpretation

Given a specification $\mathcal{K} = \{\texttt{data X where } \cdots\}$, construct a type $\texttt{ind}(\mathcal{K})$ which

1. supports constructors of the specified types,

Computational interpretation

Given a specification $\mathcal{K} = \{\texttt{data X where } \cdots\}$, construct a type $\texttt{ind}(\mathcal{K})$ which

1. supports constructors of the specified types,
2. supports an eliminator of the prescribed type,

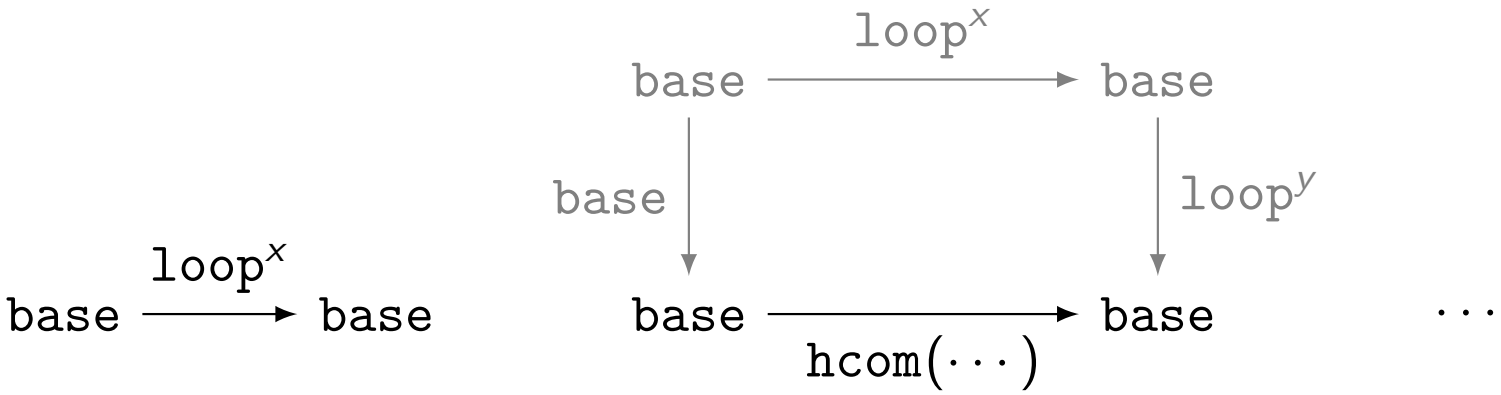
Computational interpretation

Given a specification $\mathcal{K} = \{\texttt{data X where } \cdots\}$, construct a type $\texttt{ind}(\mathcal{K})$ which

1. supports constructors of the specified types,
2. supports an eliminator of the prescribed type,
3. is Kan (supports `hcom` and `coe`).

Computational interpretation

Elements of the circle $\mathbb{S}^1$:



Computational interpretation

Q: How to represent all of these?

Computational interpretation

Q: How to represent all of these?

A: Introduce `fcom` (formal homogeneous composition) values:

$$\text{e.g.} \quad \texttt{fcom}^{0 \rightsquigarrow 1} \left(\begin{array}{ccc} y \downarrow & \bullet & \xrightarrow{\texttt{loop}^x} \bullet \\ & \texttt{base} \downarrow & \downarrow \texttt{loop}^y \\ & \bullet & \bullet \end{array} \right) \in \mathbb{S}^1$$

Computational interpretation

Q: How to represent all of these?

A: Introduce `fcom` (formal homogeneous composition) values:

$$\text{e.g.} \quad \texttt{fcom}^{0 \rightsquigarrow 1} \left(\begin{array}{ccc} y \downarrow & \bullet & \xrightarrow{\texttt{loop}^x} \bullet \\ & \texttt{base} \downarrow & \downarrow \texttt{loop}_y \\ & \bullet & \bullet \end{array} \right) \in \mathbb{S}^1$$

$$\begin{array}{lcll} \texttt{fcom}^{0 \rightsquigarrow 1}(\cdots) & \longmapsto & \texttt{base}\langle 1/y \rangle & \text{when } x = 0 \\ \texttt{fcom}^{0 \rightsquigarrow 1}(\cdots) & \longmapsto & \texttt{loop}^y \langle 1/y \rangle & \text{when } x = 1 \\ \texttt{fcom}^{0 \rightsquigarrow 1}(\cdots) & \texttt{val} & & \text{otherwise} \end{array}$$

Computational interpretation

Define the meaning of $\mathbb{S}^1$ as the least family $(\sigma_\Psi)_\Psi$ of value PERs such that

1. $\sigma_\Psi(\mathtt{base}, \mathtt{base})$
2. $\sigma_\Psi(\mathtt{loop}^x, \mathtt{loop}^x)$ for $x \in \Psi$,
3. $\sigma_\Psi(\mathtt{fcom}^{r \rightsquigarrow r'}(\sqcap), \mathtt{fcom}^{r \rightsquigarrow r'}(\sqcap'))$ whenever $\sqcap$ and $\sqcap'$ are equal open boxes according to σ.

Computational interpretation

Define the meaning of $\mathbb{S}^1$ as the least family $(\sigma_\Psi)_\Psi$ of value PERs such that

1. $\sigma_\Psi(\mathtt{base}, \mathtt{base})$
2. $\sigma_\Psi(\mathtt{loop}^x, \mathtt{loop}^x)$ for $x \in \Psi$,
3. $\sigma_\Psi(\mathtt{fcom}^{r \rightsquigarrow r'}(\sqcap), \mathtt{fcom}^{r \rightsquigarrow r'}(\sqcap'))$ whenever $\sqcap$ and $\sqcap'$ are equal open boxes according to σ.

Terms in $\mathbb{S}^1$ are terms which coherently evaluate to values in σ.

Computational interpretation

Define the meaning of $\texttt{ind}(\mathcal{K})$, where $\mathcal{K}$ is a specification in context Ψ, as the least family of value PERs $(\alpha_\psi)_{\psi:\Psi' \to \Psi}$ such that

1. $\alpha_\psi(\texttt{intro}_{\mathcal{K},i}(\cdots), \texttt{intro}_{\mathcal{K},i}(\cdots'))$ when their arguments are equal "according to α"

Computational interpretation

Define the meaning of $\mathtt{ind}(\mathcal{K})$, where $\mathcal{K}$ is a specification in context Ψ, as the least family of value PERs $(\alpha_\psi)_{\psi:\Psi'\to\Psi}$ such that

1. $\alpha_\psi(\mathtt{intro}_{\mathcal{K},i}(\cdots), \mathtt{intro}_{\mathcal{K},i}(\cdots'))$ when their arguments are equal "according to α"
   - For equality of recursive arguments, define an interpretation of C $\mathtt{atype}$ at α:

$$\begin{array}{rcl} \{\!|\mathtt{X}|\!\}(\alpha) & := & \alpha \\ \{\!|(b{:}B) \to \mathrm{C}|\!\}(\alpha) & := & \mathrm{PI}([\![B]\!], \{\!|\mathrm{C}|\!\}(\alpha)) \end{array}$$

Computational interpretation

Define the meaning of $\mathtt{ind}(\mathcal{K})$, where $\mathcal{K}$ is a specification in context Ψ, as the least family of value PERs $(\alpha_\psi)_{\psi:\Psi'\to\Psi}$ such that

1. $\alpha_\psi(\mathtt{intro}_{\mathcal{K},i}(\cdots), \mathtt{intro}_{\mathcal{K},i}(\cdots'))$ when their arguments are equal "according to α"
   - For equality of recursive arguments, define an interpretation of C atype at α:

$$\begin{array}{rcl} \{\!|\mathtt{X}|\!\}(\alpha) & := & \alpha \\ \{\!|(b{:}B) \to \mathrm{C}|\!\}(\alpha) & := & \mathrm{PI}([\![B]\!], \{\!|\mathrm{C}|\!\}(\alpha)) \end{array}$$

   - Operational semantics of intro terms is defined by the spec:

$$\begin{array}{lcll} \mathtt{intro}_{\mathcal{K},1}(\cdots) & \longmapsto & \mathrm{M}_j(\cdots) & \text{when } j \text{ least such that } \xi_j \\ \mathtt{intro}_{\mathcal{K},1}(\cdots) & \mathtt{val} & & \text{otherwise} \end{array}$$

Computational interpretation

Define the meaning of $\mathtt{ind}(\mathcal{K})$, where $\mathcal{K}$ is a specification in context Ψ, as the least family of value PERs $(\alpha_\psi)_{\psi:\Psi'\to\Psi}$ such that

1. $\alpha_\psi(\mathtt{intro}_{\mathcal{K},i}(\cdots), \mathtt{intro}_{\mathcal{K},i}(\cdots'))$ when their arguments are equal "according to α"
   - For equality of recursive arguments, define an interpretation of C $\mathtt{atype}$ at α:

$$\begin{array}{rcl} \{\!|\mathtt{X}|\!\}(\alpha) & := & \alpha \\ \{\!|(b{:}B) \to \mathrm{C}|\!\}(\alpha) & := & \mathrm{PI}([\![B]\!], \{\!|\mathrm{C}|\!\}(\alpha)) \end{array}$$

   - Operational semantics of $\mathtt{intro}$ terms is defined by the spec:

$$\begin{array}{lcll} \mathtt{intro}_{\mathcal{K},1}(\cdots) & \longmapsto & \mathrm{M}_j(\cdots) & \text{when } j \text{ least such that } \xi_j \\ \mathtt{intro}_{\mathcal{K},1}(\cdots) & \mathtt{val} & & \text{otherwise} \end{array}$$

2. $\alpha_\psi(\mathtt{fcom}^{r\rightsquigarrow r'}(\sqcap), \mathtt{fcom}^{r\rightsquigarrow r'}(\sqcap'))$ whenever $\sqcap$ and $\sqcap'$ are equal open boxes according to α.

Implementing the Kan conditions

Implementing the Kan conditions

1. `hcom` is easy:

$$\texttt{hcom}_{\texttt{ind}(\mathcal{K})} \longmapsto \texttt{fcom}$$

Implementing the Kan conditions

1. `hcom` is easy:

$$\texttt{hcom}_{\texttt{ind}(\mathcal{K})} \longmapsto \texttt{fcom}$$

2. What about `coe`?

Implementing the Kan conditions

1. `hcom` is easy:

$$\texttt{hcom}_{\texttt{ind}(\mathcal{K})} \longmapsto \texttt{fcom}$$

2. What about `coe`?
   - To coerce an `intro`, apply the appropriate coercion to each argument

Implementing the Kan conditions

1. `hcom` is easy:

$$\texttt{hcom}_{\texttt{ind}(\mathcal{K})} \longmapsto \texttt{fcom}$$

2. What about `coe`?
   - To coerce an `intro`, apply the appropriate coercion to each argument
   - (For higher constructors, "boundary correction" is required to ensure coherence. This is achieved using `fcom`.)

Implementing the Kan conditions

1. `hcom` is easy:

$$\texttt{hcom}_{\texttt{ind}(\mathcal{K})} \longmapsto \texttt{fcom}$$

2. What about `coe`?
   - To coerce an `intro`, apply the appropriate coercion to each argument
   - (For higher constructors, "boundary correction" is required to ensure coherence. This is achieved using `fcom`.)
   - To coerce an `fcom`, apply coercion to each face of the box

Implementing the Kan conditions

1. `hcom` is easy:

$$\texttt{hcom}_{\texttt{ind}(\mathcal{K})} \longmapsto \texttt{fcom}$$

2. What about `coe`?
   - To coerce an `intro`, apply the appropriate coercion to each argument
   - (For higher constructors, "boundary correction" is required to ensure coherence. This is achieved using `fcom`.)
   - To coerce an `fcom`, apply coercion to each face of the box

Implementing the Kan conditions

1. `hcom` is easy:

$$\mathtt{hcom}_{\mathtt{ind}(\mathcal{K})} \longmapsto \mathtt{fcom}$$

2. What about `coe`?
   - To coerce an `intro`, apply the appropriate coercion to each argument
   - (For higher constructors, "boundary correction" is required to ensure coherence. This is achieved using `fcom`.)
   - To coerce an `fcom`, apply coercion to each face of the box

Kan conditions are achieved by a fiberwise fibrant replacement

- The construction of $\mathtt{ind}(\mathcal{K})$ from $\mathcal{K}$ is parametric in the ambient dimension context Ψ
- Add solutions to boxes in each fiber, and get the rest from the Kan structure of the component types

Properties of the computational interpretation

1. Introduction rules:

Properties of the computational interpretation

1. Introduction rules:
   - Well-formed constructor terms are in $\texttt{ind}(\mathcal{K})$,

Properties of the computational interpretation

1. Introduction rules:
   - Well-formed constructor terms are in $\texttt{ind}(\mathcal{K})$,
   - and have the boundary they were prescribed.

Properties of the computational interpretation

1. Introduction rules:
   - Well-formed constructor terms are in $\texttt{ind}(\mathcal{K})$,
   - and have the boundary they were prescribed.
2. Elimination rules:

Properties of the computational interpretation

1. Introduction rules:
   - Well-formed constructor terms are in $\texttt{ind}(\mathcal{K})$,
   - and have the boundary they were prescribed.
2. Elimination rules:
   - The eliminator can map from $\texttt{ind}(\mathcal{K})$ into any Kan type,

Properties of the computational interpretation

1. Introduction rules:
   - Well-formed constructor terms are in $\texttt{ind}(\mathcal{K})$,
   - and have the boundary they were prescribed.
2. Elimination rules:
   - The eliminator can map from $\texttt{ind}(\mathcal{K})$ into any Kan type,
   - and satisfies β-rules up to exact equality.

Canonicity

Canonicity

1. For strict booleans:
   - Same as before: if $M \in \texttt{bool}$ then $M \Downarrow \texttt{true}$ or $M \Downarrow \texttt{false}$.

Canonicity

1. For strict booleans:
   - Same as before: if $M \in \texttt{bool}$ then $M \Downarrow \texttt{true}$ or $M \Downarrow \texttt{false}$.
2. For inductive types:
   - Every element of $\texttt{ind}(\mathcal{K})$ evaluates to a canonical value of $\texttt{ind}(\mathcal{K})$...

Canonicity

1. For strict booleans:
   - Same as before: if $M \in \texttt{bool}$ then $M \Downarrow \texttt{true}$ or $M \Downarrow \texttt{false}$.
2. For inductive types:
   - Every element of $\texttt{ind}(\mathcal{K})$ evaluates to a canonical value of $\texttt{ind}(\mathcal{K})$...
   - i.e., an `intro` or `fcom` value.

Canonicity

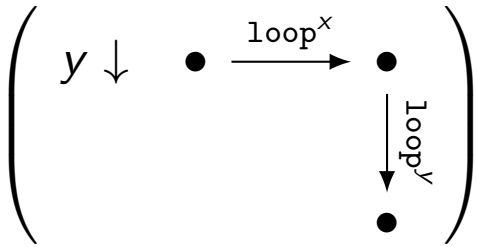
1. For strict booleans:
   - Same as before: if $M \in \texttt{bool}$ then $M \Downarrow \texttt{true}$ or $M \Downarrow \texttt{false}$.
2. For inductive types:
   - Every element of $\texttt{ind}(\mathcal{K})$ evaluates to a canonical value of $\texttt{ind}(\mathcal{K})$...
   - i.e., an `intro` or `fcom` value.

Canonicity

1. For strict booleans:
   - Same as before: if $M \in \texttt{bool}$ then $M \Downarrow \texttt{true}$ or $M \Downarrow \texttt{false}$.
2. For inductive types:
   - Every element of $\texttt{ind}(\mathcal{K})$ evaluates to a canonical value of $\texttt{ind}(\mathcal{K})$...
   - i.e., an `intro` or `fcom` value.
3. Improvements:
   - Exclude `fcom` values at dimension 0:

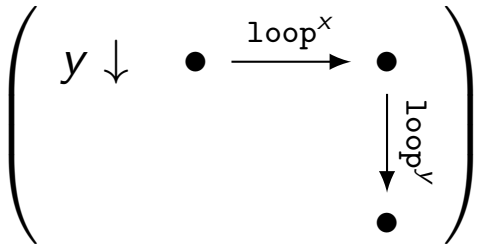
Canonicity

1. For strict booleans:
   - Same as before: if $M \in \texttt{bool}$ then $M \Downarrow \texttt{true}$ or $M \Downarrow \texttt{false}$.
2. For inductive types:
   - Every element of $\texttt{ind}(\mathcal{K})$ evaluates to a canonical value of $\texttt{ind}(\mathcal{K})$...
   - i.e., an `intro` or `fcom` value.
3. Improvements:
   - Exclude `fcom` values at dimension 0:
     - ★ Adjust the Kan condition: only require `hcom`s for which every closing instance reduces. This suffices!

BAD: $\texttt{fcom}^{0 \rightsquigarrow 1}$ 

Canonicity

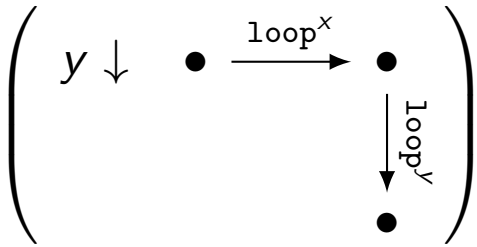
1. For strict booleans:
   - Same as before: if $M \in \texttt{bool}$ then $M \Downarrow \texttt{true}$ or $M \Downarrow \texttt{false}$.
2. For inductive types:
   - Every element of $\texttt{ind}(\mathcal{K})$ evaluates to a canonical value of $\texttt{ind}(\mathcal{K})$...
   - i.e., an `intro` or `fcom` value.
3. Improvements:
   - Exclude `fcom` values at dimension 0:
     - ★ Adjust the Kan condition: only require `hcom`s for which every closing instance reduces. This suffices!

BAD: $\texttt{fcom}^{0 \rightsquigarrow 1}$ 

     - ★ Then, every 0-dimensional element of $\texttt{ind}(\mathcal{K})$ evaluates to an `intro` value.

Canonicity

1. For strict booleans:
   - Same as before: if $M \in \texttt{bool}$ then $M \Downarrow \texttt{true}$ or $M \Downarrow \texttt{false}$.
2. For inductive types:
   - Every element of $\texttt{ind}(\mathcal{K})$ evaluates to a canonical value of $\texttt{ind}(\mathcal{K})$...
   - i.e., an `intro` or `fcom` value.
3. Improvements:
   - Exclude `fcom` values at dimension 0:
     - ★ Adjust the Kan condition: only require `hcom`s for which every closing instance reduces. This suffices!

BAD: $\texttt{fcom}^{0 \rightsquigarrow 1}$ 

     - ★ Then, every 0-dimensional element of $\texttt{ind}(\mathcal{K})$ evaluates to an `intro` value.
   - Implement `hcom` directly for 0-dimensional inductive types.

Type systems & Universes

Type systems & Universes

1. At each stage of construction, add the inductive type for every currently well-typed specification

Type systems & Universes

1. At each stage of construction, add the inductive type for every currently well-typed specification
2. The fixed-point system is then closed under all inductive types

Type systems & Universes

1. At each stage of construction, add the inductive type for every currently well-typed specification
2. The fixed-point system is then closed under all inductive types
3. We have parameterized inductives "for free"

Indexed inductive types

Indexed inductive types

- A indexed family of types which are simultaneously inductively generated by constructors

$$\begin{aligned}&\texttt{data}\ \texttt{Id}_A : A \to A \to \texttt{type}\ \texttt{where}\\ &\quad \texttt{refl}(a) \in (a : A) \to \texttt{Id}_A(a, a)\end{aligned}$$

Indexed inductive types

- A indexed family of types which are simultaneously inductively generated by constructors

$$\begin{aligned}&\texttt{data}\ \texttt{Id}_A : A \to A \to \texttt{type}\ \texttt{where}\\ &\quad \texttt{refl}(a) \in (a : A) \to \texttt{Id}_A(a, a)\end{aligned}$$

Indexed inductive types

- A indexed family of types which are simultaneously inductively generated by constructors

$$\begin{aligned}&\texttt{data}\ \texttt{Id}_A : A \to A \to \texttt{type}\ \texttt{where}\\ &\quad \texttt{refl}(a) \in (a : A) \to \texttt{Id}_A(a, a)\end{aligned}$$

- Even with 0-constructors, need new elements to satisfy the Kan condition

$$\texttt{coe}^{0 \rightsquigarrow y}_{x.\texttt{Id}_{\mathbb{S}1}(\texttt{base},\texttt{loop}^x)}(\texttt{refl}(\texttt{base})) \longmapsto ???$$

Indexed inductive types

Suppose

$$A\ \texttt{type} \qquad B\ \texttt{type} \qquad F \in A \to B$$

Want to define

$$\begin{aligned}&\texttt{data}\ \texttt{fib}(A; B; F) : B \to \texttt{type}\ \texttt{where}\\ &\quad \texttt{refl}(a) \in (a : A) \to \texttt{fib}(A; B; F)(F(a))\end{aligned}$$

Indexed inductive types

Suppose

$$A\ \texttt{type} \qquad B\ \texttt{type} \qquad F \in A \to B$$

Want to define

$$\begin{aligned}&\texttt{data}\ \texttt{fib}(A; B; F) : B \to \texttt{type}\ \texttt{where}\\ &\quad \texttt{refl}(a) \in (a : A) \to \texttt{fib}(A; B; F)(F(a))\end{aligned}$$

- Recall: for ordinary CITs, only needed free `hcom` values
- For indexed inductive types, also need `coe` values – but only for coercing between indices

Indexed inductive types

Suppose

$$A\ \mathtt{type} \qquad B\ \mathtt{type} \qquad F \in A \to B$$

Want to define

$$\begin{aligned}&\mathtt{data}\ \mathtt{fib}(A; B; F) : B \to \mathtt{type}\ \mathtt{where}\\ &\quad \mathtt{refl}(a) \in (a : A) \to \mathtt{fib}(A; B; F)(F(a))\end{aligned}$$

- Recall: for ordinary CITs, only needed free `hcom` values
- For indexed inductive types, also need `coe` values – but only for coercing between indices

$$\mathtt{fcoe}^{r \rightsquigarrow r'}_{x.N} \quad : \quad \mathtt{fib}(A; B; F)(N\langle r/x\rangle) \to \mathtt{fib}(A; B; F)(N\langle r'/x\rangle)$$

Indexed inductive types

Suppose

$$A\ \mathtt{type} \qquad B\ \mathtt{type} \qquad F \in A \to B$$

Want to define

$$\begin{array}{l}\mathtt{data}\ \mathtt{fib}(A; B; F) : B \to \mathtt{type}\ \mathtt{where} \\ \quad \mathtt{refl}(a) \in (a : A) \to \mathtt{fib}(A; B; F)(F(a))\end{array}$$

- Recall: for ordinary CITs, only needed free `hcom` values
- For indexed inductive types, also need `coe` values – but only for coercing between indices

$$\mathtt{fcoe}^{r \rightsquigarrow r'}_{x.N} \quad : \quad \mathtt{fib}(A; B; F)(N\langle r/x\rangle) \to \mathtt{fib}(A; B; F)(N\langle r'/x\rangle)$$

$$\mathtt{tcoe}^{r \rightsquigarrow r'}_{x.(A;B;F)} \quad : \quad \mathtt{fib}(A; B; F)\langle r/x\rangle(N) \to \mathtt{fib}(A; B; F)\langle r'/x\rangle(\qquad)$$

Indexed inductive types

Suppose

$$A\ \mathtt{type} \qquad B\ \mathtt{type} \qquad F \in A \to B$$

Want to define

$$\begin{array}{l}\mathtt{data}\ \mathtt{fib}(A; B; F) : B \to \mathtt{type}\ \mathtt{where} \\ \quad \mathtt{refl}(a) \in (a : A) \to \mathtt{fib}(A; B; F)(F(a))\end{array}$$

- Recall: for ordinary CITs, only needed free `hcom` values
- For indexed inductive types, also need `coe` values – but only for coercing between indices

$$\mathtt{fcoe}^{r \rightsquigarrow r'}_{x.N} \quad : \quad \mathtt{fib}(A; B; F)(N\langle r/x\rangle) \to \mathtt{fib}(A; B; F)(N\langle r'/x\rangle)$$

$$\mathtt{tcoe}^{r \rightsquigarrow r'}_{x.(A;B;F)} \quad : \quad \mathtt{fib}(A; B; F)\langle r/x\rangle(N) \to \mathtt{fib}(A; B; F)\langle r'/x\rangle(\mathtt{coe}^{r \rightsquigarrow r'}_{x.B}(N))$$

Indexed inductive types

- Define `fib`$(A; B; F)$ as composed of `refl`, `fcoe`, and `fcom` terms
- Implement `tcoe` using the Kan structure on A and B
- Implement `coe` using `fcoe` and `tcoe`

Indexed inductive types

- Define `fib`$(A; B; F)$ as composed of `refl`, `fcoe`, and `fcom` terms
- Implement `tcoe` using the Kan structure on A and B
- Implement `coe` using `fcoe` and `tcoe`

- To eliminate into a Kan type, need only supply the `refl` case
- Exact β-rule for `refl`! (see also [Swa14; Coh+15])

The future

- Indexed inductive types in generality
- Inductive-inductive types
- Implementation in RedPRL

References

Evan Cavallo and Robert Harper. *Computational Higher Type Theory IV: Inductive Types*. arXiv:1801.01568.

Evan Cavallo and Robert Harper. *Higher Inductive Types in Cubical Computational Type Theory*. Preprint.

THANKS