

International Conference on Data Science and Intelligent Analysis of Information 2018

Constructive Proofs of Heterogeneous Equalities in Cubical Type Theory

Scientific and Technical Library of the Igor Sikorsky Kyiv Polytechnic Institute, Kyiv, Ukraine.
ICDSIAI 2018 Maksym Sokhatskyi, Pavlo Maslianko

Abstract

Cubical Base Library

We demonstrate the usage of part of base library by showing how to build a constructive proof of heterogeneous equality, the simple and elegant formulation of the equality problem, that was impossible to achieve in pure Martin-Löf Type Theory (MLTT). The machinery used in this article unveils the internal aspect of path equalities and isomorphism, used e.g. for proving univalence axiom, that became possible only in CTT.

Structure of Presentation

From proving to Extraction, Linking and Running

1) Type Theory

- Pi Type
- Sigma Type
- Equ Type

2) Equality Problem

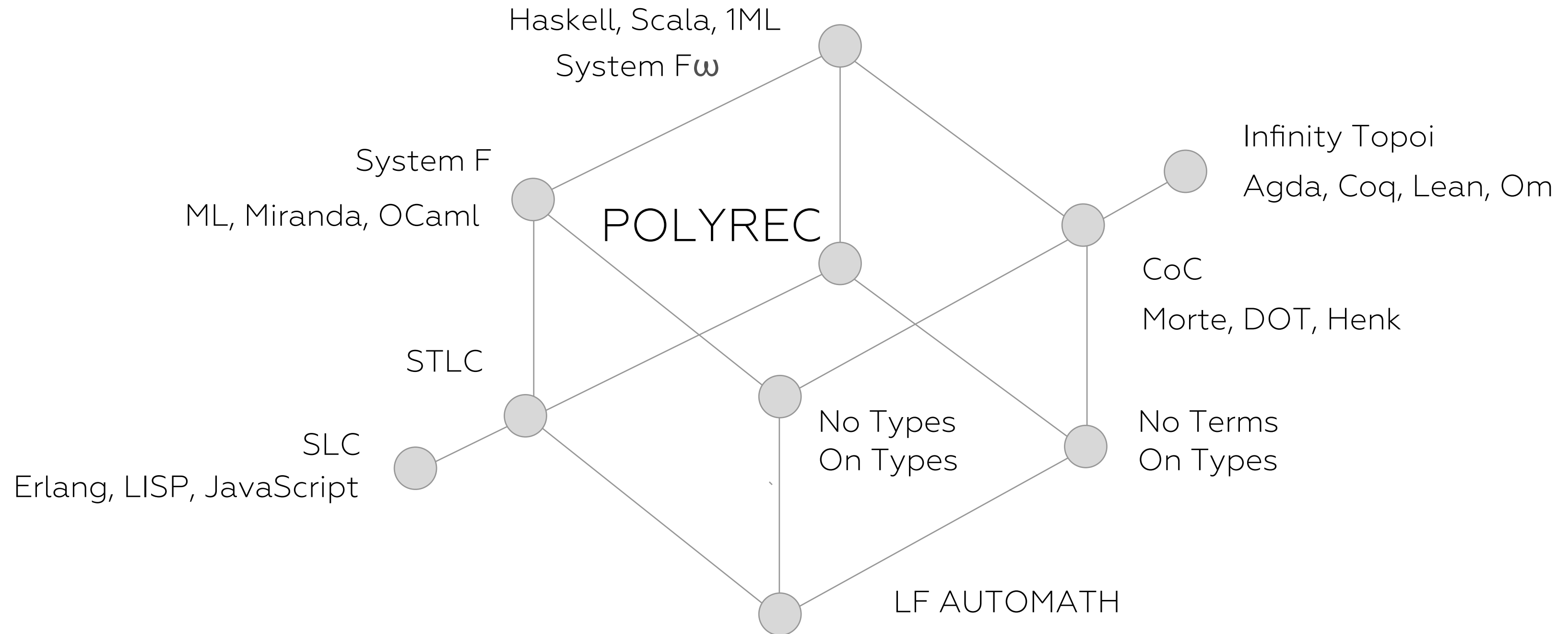
- Definitional Equality
- Propositional Equality
- Heterogeneous and Globular Equality

3) Computational Semantics — Cubical Type Theory 2017

- CCHM Fibrations
- Homotopy Path Type as Equality with de Morgan Algebra
- Composition, Glueing, Filling
- Cubical Base Library and further work

Programs as Proofs

in Extended Lambda Cube



Model Verification Process

From proving to Extraction, Linking and Running

1) Models

- Real Analysis
- Abstract Algebra
- Homotopy Theory

3) Extraction

- LLVM
- Interpreters
- Detying
- Optimization
- Linking

2) Core — Infinity Language

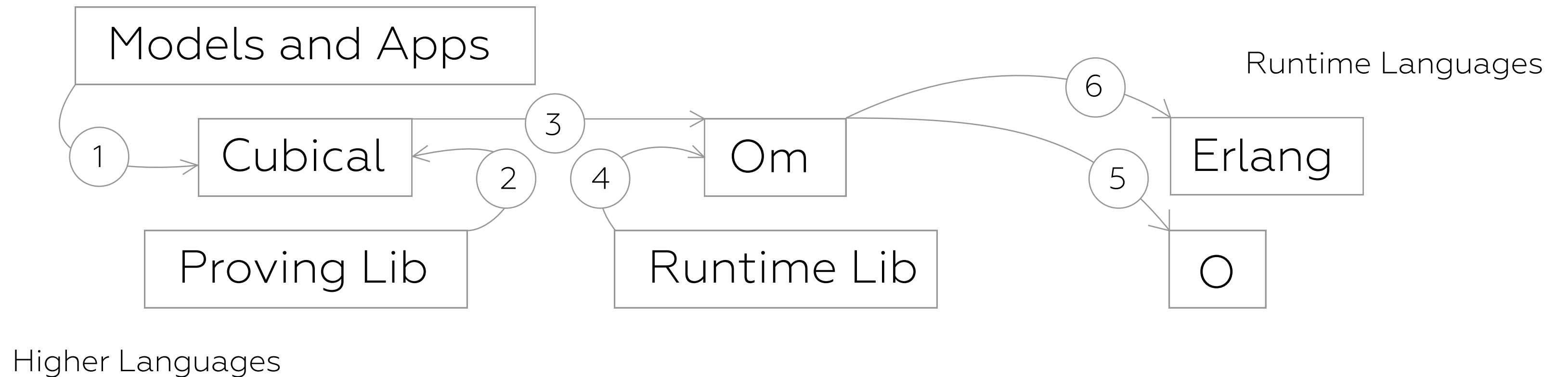
- Model Verification
- Normalization
- Bidirectional Checking
- Pure Type System (Om)
- Identity
- Induction
- Homotopy Interval $[0,1]$

4) Runtimes

- O
- Erlang
- V8
- JVM

Prover Structure

Models, Languages, Libraries, Applications



[3,4] cover the presented work, [1,2,5,6] cover the future works.

MLTT 1984

Type Theory as new Foundations of Mathematics

$U_0 : U_1 : U_2 : U_3 : \dots \infty$ — universes

$x : A$ — x is a point in space A

$y = [x : A]$ — x and y are definitionally equal objects of type A

Π

Σ

$=$

1. Formation Rules

$x:A \rightarrow B(x)$

$x:A * B(x)$

$x:A = y:A$

2. Introduction Rules

$\lambda (x: A) \rightarrow B(x)$

$(x, B(x))$

$\text{refl } A \ x$

3. Elimination Rules

$f \ a = B(a)$

$\text{pr1}, \text{pr2}$

J

4. Computational Rules

Equalities

Equalities

Equalities

Pi Type

Inductive Type, AST, Logical Framework

```
data pts
  = star (n: nat)
  | var (x: name) (l: nat)
  | pi (x: name) (l: nat) (d c: lang)
  | lambda (x: name) (l: nat) (d c: lang)
  | app (f a: lang)
```

$\langle \rangle ::= \#option$

$T ::= \#identifier$

$U ::= * \langle \#number \rangle$

$O ::= U \mid T \mid (O) \mid O O \mid O \rightarrow O$
 $\mid \backslash (l: O) \rightarrow O \mid (l: O) \rightarrow O$

$\Pi (A:U)(P:A \rightarrow U):U = (x:A) \rightarrow P(x)$

$\lambda (A : U) (B: A \rightarrow U) (a : A) (b: B a): A \rightarrow B a = \backslash (x: A) \rightarrow b$

$\text{app } (A : U) (B: A \rightarrow U) (a : A) (f: A \rightarrow B a): B a = f a$

Sigma Type

Inductive Type, AST, Logical Framework

data exists

= sigma (n: name) (a b: lang)
| pair (a b: lang)
| fst (p: lang)
| snd (p: lang)

O := **Σ** (x: O), O | (O,O) | O.1 | O.2

Sigma (A : U) (B : A -> U) : U = (x : A) * B x

pair (A : U) (B: A -> U) (a : A) (b: B a): Sigma A B = (a,b)

pr1 (A: U) (B: A -> U) (x: Sigma A B): A = x.1

pr2 (A: U) (B: A -> U) (x: Sigma A B): B (pr1 A B x) = x.2

Sigma Types in Pi

Typing and Introduction Rules

-- Sigma/@

\ (A: *)

-> \ (P: A -> *)

-> \ (n: A)

-> \ (Exists: *)

-> \ (Intro: A -> P n -> Exists)

-> Exists

-- Sigma/Intro

\ (A: *)

-> \ (P: A -> *)

-> \ (x: A)

-> \ (y: P x)

-> \ (Exists: *)

-> \ (Intro: \ (x:A) -> P x -> Exists)

-> Intro x y

Sigma Types in Pi

Dependent Elimimators

-- Sigma/fst

\ (A: *)

-> \ (B: A -> *)

-> \ (n: A)

-> \ (S: #Sigma/@ A B n)

-> S A (\ (x: A) -> \ (y: B n) -> x)

-- Sigma/snd

\ (A: *)

-> \ (B: A -> *)

-> \ (n: A)

-> \ (S: #Sigma/@ A B n)

-> S (B n) (\ (: A) -> \ (y: B n) -> y)

Equ Types in Any ?

```
-- Equ/@  
\ (A: *)  
-> \ (x: A)  
-> \ (y: A)  
-> \ (Equ: A -> A -> *)  
-> \ (Refl: \ (z: A) -> Equ z z) -> Equ x y
```

```
-- Equ/Refl \ (A: *)  
-> \ (x: A)  
-> \ (Equ: A -> A -> *)  
-> \ (Refl: \ (z: A) -> Equ z z)  
-> Refl x
```

```
-- Equ/J  
???
```

Definitional Equality

Definitional, built into Type Checker

eq (:star ,N)	(:star ,N)	→ true
(:var,N,l)	(:var,(N,l))	→ true
(:remote ,N)	(:remote ,N)	→ true
(:pi,N1,O,l1,O1)	(:pi,N2,O,l2,O2)	→ let :true = eq l1 l2 in eq O1 (subst (shift O2 N1 O) N2 (:var,N1,O) O)
(:fn,N1,O,l1,O1)	(:fn,N2,O,l2,O2)	→ let :true = eq l1 l2 in eq O1 (subst (shift O2 N1 O) N2 (:var,N1,O) O)
(:app,F1,A1)	(:app,F2,A2)	→ let :true = eq F1 F2 in eq A1 A2
(A,	B)	→ (:error ,(:eq,A,B))

Propositional Logic

According to Brouwer–Heyting–Kolmogorov interpretation

Π

Σ

$=$

0

1

$+$

$x:A \rightarrow B(x)$

$\lambda (x: A) \rightarrow B(x)$

$f\ a = B(a)$

$x:A * B(x)$

$(x, B(x))$

$pr1, pr2$

$x:A = y:A$

$refl\ A\ x$

J

data empty

$elim0$

data unit

$= tt$

$elim1$

data either (A B :: U)

$= inl\ (a: A) \mid inr\ (b: B)$

$elimEither$

Bishop's Constructive Analysis

Reflexivity, Transitivity, Symmetry

Setoid $(A: U): U$

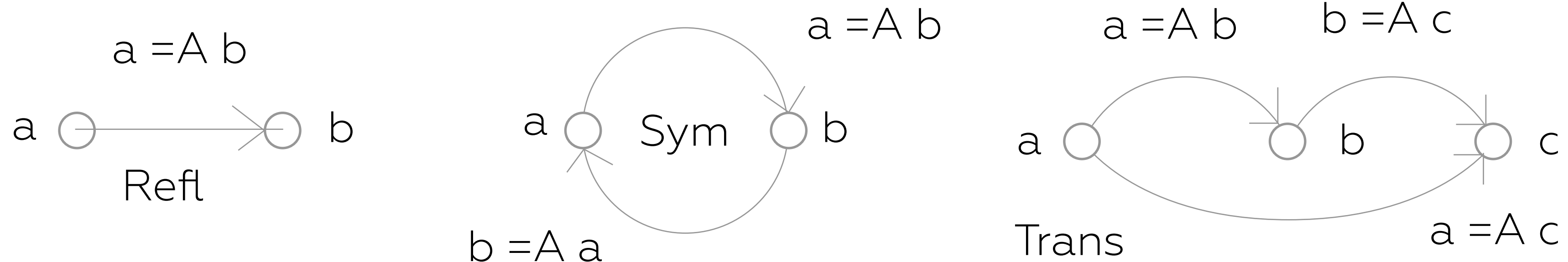
= (Carrier: A)

* (Equ: $(a\ b: A) \rightarrow \text{Path } A\ a\ b$)

* (Refl: $(x: A) \rightarrow \text{Equ } x\ x$)

* (Trans: $(x_1, x_2, x_3: A) \rightarrow \text{Equ } x_1\ x_2 \rightarrow \text{Equ } x_2\ x_3 \rightarrow \text{Equ } x_1\ x_3$)

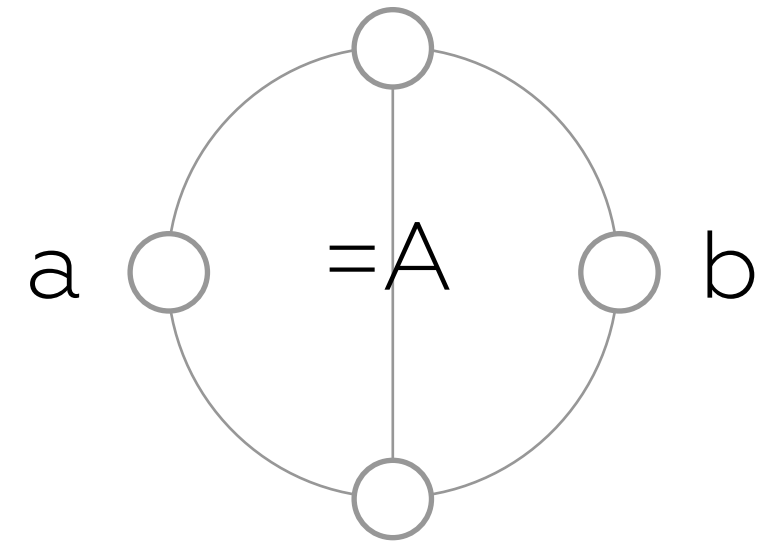
* (Sym: $(x_1, x_2: A) \rightarrow \text{Equ } x_1\ x_2 \rightarrow \text{Equ } x_2\ x_1$)



Globular Theory

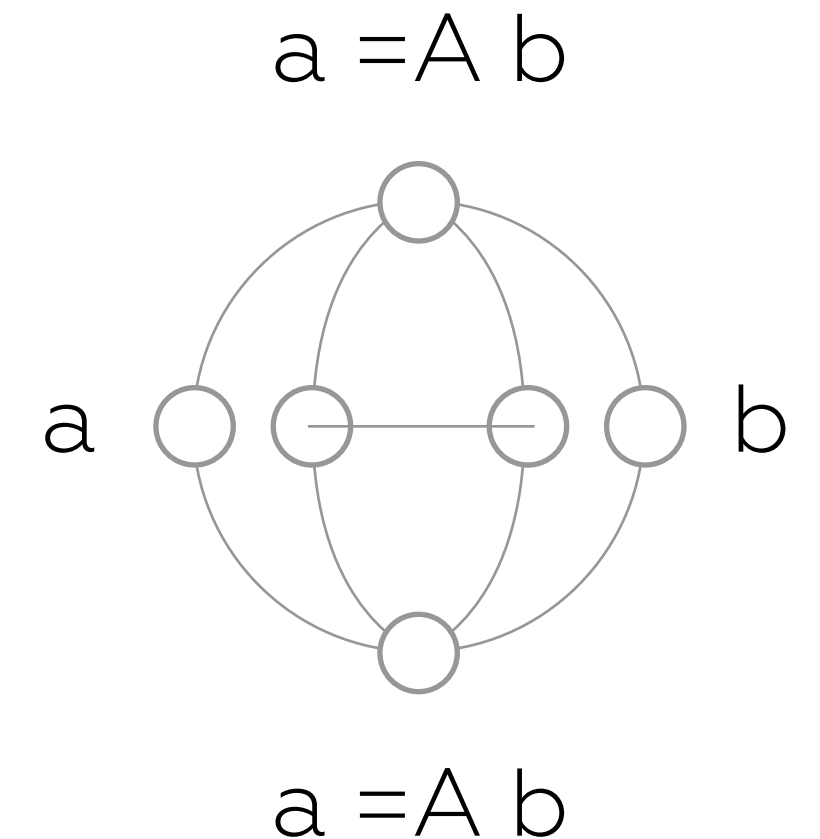
Multidimensional Equality

$$a =_A b$$



$$((a =_A b) =_{(=A)} (a =_A b))$$

$$((a =_A b) =_{(=A)} (a =_A b)) =_{(=A)} ((a =_A b) =_{(=A)} (a =_A b))$$



Equ Type a la Martin-Löf

Path (A: U) (a b: A): U = axiom

HeteroEqu (A B: U) (a: A) (b: B) (P: Path U A B) : U = axiom — PathP P a b

Equ (A: U) (x y: A): U = HeteroEqu A A x y (<i>A)

reflect (A: U) (a: A): Equ A a a = <i> a

D (A: U) : U = (x y: A) -> Equ A x y -> U

singl (A: U) (a: A): U = (x : A) * Equ A a x

eta (A: U) (a: A): singl A a = (a,reflect A a)

cong (A B: U) (f: A->B) (a b: A) (p: Equ A a b): Equ B (f a) (f b)

subst (A: U) (P: A->U) (a b: A) (p: Equ A a b) (e: P a): P b

J (A: U) (a: A) (C: (x : A) -> Path A a x -> U)
(d: C a (refl A a)) (x: A) (p: Path A a x): C x p

Path Types as Cubes

Syntax and Model

```
data hts
  = path (a b: lang)
  | path_lam (n: name) (a b: lang)
  | path_app (f: name) (a b: lang)
  | comp_ (a b: lang)
  | fill_ (a b c: lang)
  | glue_ (a b c: lang)
  | glue_elem (a b: lang)
  | unglue_elem (a b: lang)
```

$p : [\text{PathP } A \ a \ b, p = (i: I) \rightarrow A]$

$a : A \quad \bigcirc \longrightarrow \bigtriangleright \bigcirc \quad b : A$

de Morgan: $1 - i \mid i \mid i \wedge j \mid i \vee j$

FunExt

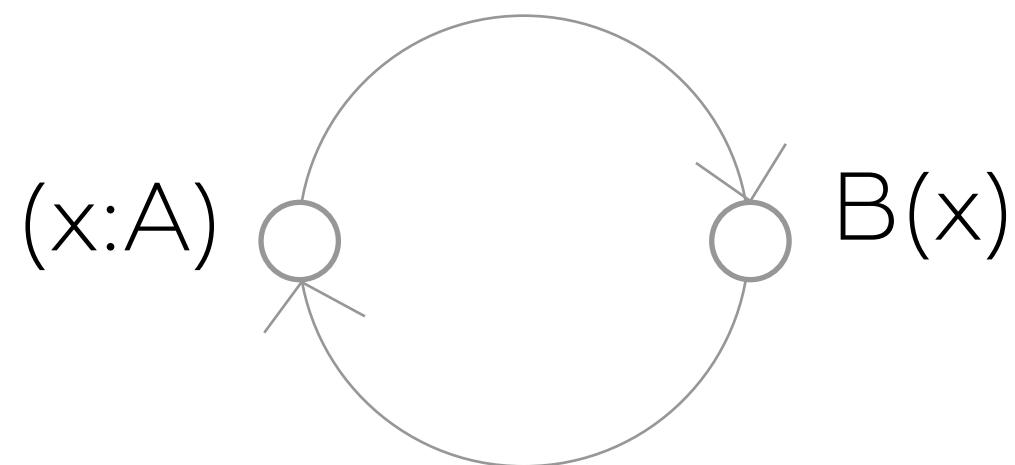
Syntax and Model

```

piExt (A: U) (B: A -> U) (f g: (x:A) -> B x)
      (p: (x:A) -> Path (B x) (f x) (g x))
      : Path ((y:A) -> B y) f g
= <i> \ (a: A) -> (p a) @ i


```

$f : (x:A) \rightarrow B(x)$



$g : (x:A) \rightarrow B(x)$

$f = (A \rightarrow B) \ g$

$f : A \rightarrow B$  $g : A \rightarrow B$

$<i> \backslash (a:A) \rightarrow p \ a \ @ \ i$

h-Types

```
data N = Z | S (n: N)
n_grpd (A: U) (n: N): U = (a b: A) -> rec A a b n where
  rec (A: U) (a b: A) : (k: N) -> U
    = split { Z -> Path A a b ; S n -> n_grpd (Path A a b) n }
```

```
inf_grpd (A: U): U = (carrier: A) * (eq: (a b: A) -> Path A a b)
                                     * ((a b: A) -> inf_grpd (Path A a b))
```

```
isContr (A: U): U = (x: A) * ((y: A) -> Path A x y)
```

```
isProp (A: U): U = n_grpd A Z
```

```
isSet (A: U): U = n_grpd A (S Z)
```

```
isGroupoid (A: U): U = n_grpd A (S (S Z))
```

```
PROP : U = (X:U) * isProp X
```

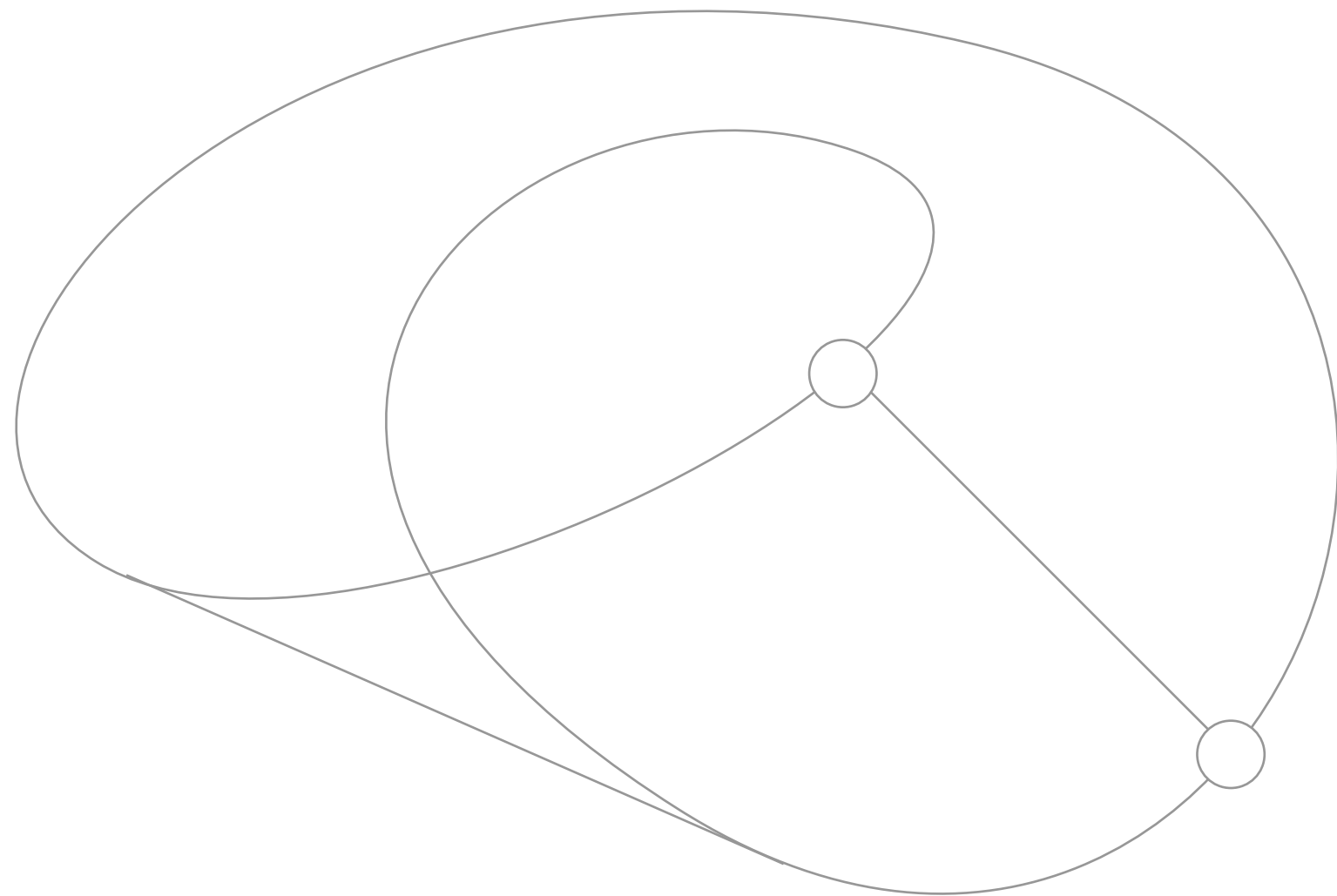
```
SET : U = (X:U) * isSet X
```

```
GROUPOID : U = (X:U) * isGroupoid X
```

```
INF_GROUPOID : U = (X:U) * isInfinityGroupoid X
```

Weak Equivalence

$\text{fiber } (A \ B: U) \ (f: A \rightarrow B) \ (y: B): U = (x: A) * \text{Path } B \ y \ (f \ x)$
 $\text{isEquiv } (A \ B: U) \ (f: A \rightarrow B): U = (y: B) \rightarrow \text{isContr } (\text{fiber } A \ B \ f \ y)$
 $\text{equiv } (A \ B: U): U = (f: A \rightarrow B) * \text{isEquiv } A \ B \ f$



Fiber Bundle: $F \rightarrow E \rightarrow B$

Moebius $E = S^1 * [0,1] = \text{total}$

Trivial: $E = B * F$

$p : \text{total} \rightarrow B$

$\text{fiber} : B \rightarrow \text{total}$

$\text{total} = (y: B) * \text{fiber}(y)$

Fiber= $\Pi (B: U) (F: B \rightarrow U) (y: B)$

$: \text{Path } U \ (\text{fiber } (\text{total } B \ F) \ B \ (\text{trivial } B \ F) \ y) \ (F \ y)$

Isomorphism

```
isIso (A B: U): U
= (f: A -> B)
* (g: B -> A)
* (s: section A B f g)
* (t: retract A B f g)
* unit
```

```
iso: U
= (A: U)
* (B: U)
* isIso A B
```

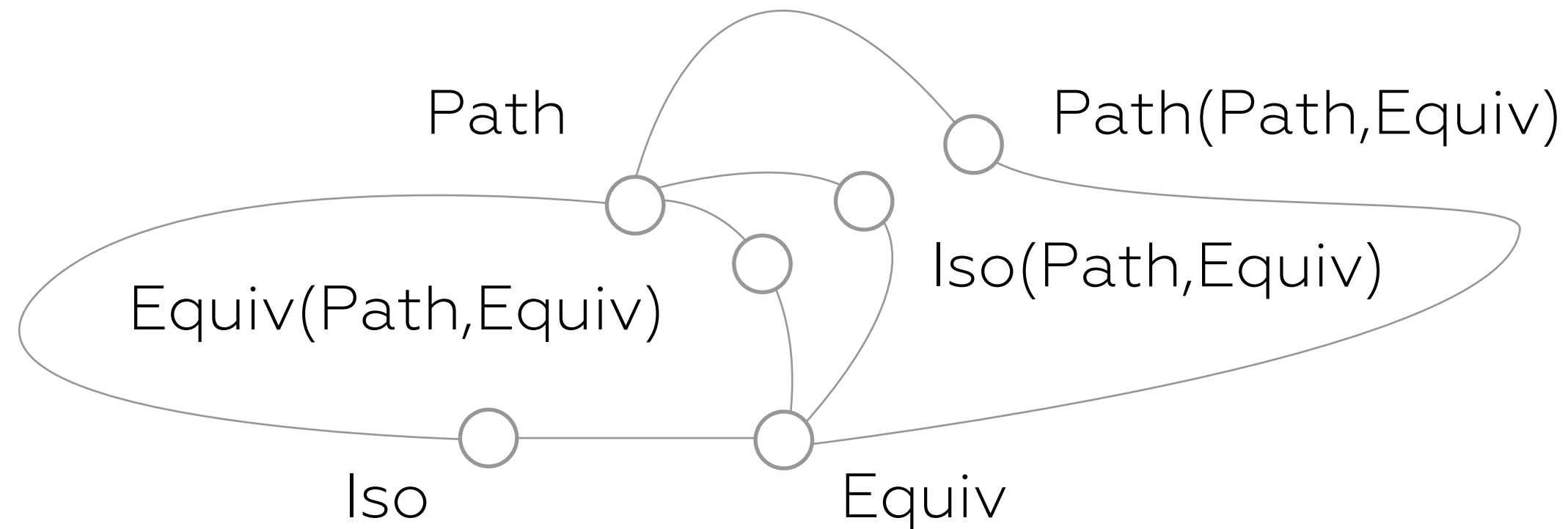
```
isoPath (A B: U) (f: A -> B) (g: B -> A)
(s: section A B f g) (t: retract A B f g): Path U A B
= <i> Glue B [ (i = 0) -> (A,f,isoToEquiv A B f g s t),
(i = 1) -> (B,idfun B,idIsEquiv B) ]
```

```
isoToPath (i: iso): Path U i.1 i.2.1
= isoPath i.1 i.2.1 i.2.2.1 i.2.2.2.1 i.2.2.2.2.1 i.2.2.2.2.2.1
```

```
section (A B: U) (f: A -> B) (g: B -> A): U = (b: B) -> Path B (f (g b)) b
retract (A B: U) (f: A -> B) (g: B -> A): U = (a: A) -> Path A (g (f a)) a
```

Univalence Axiom

All Equalities Should Be Equal



```

lemIso (A B: U) (f: A -> B) (g: B -> A) (s: section A B f g) (t: retract A B f g)
  (y: B) (x0 x1: A) (p0: Path B y (f x0)) (p1: Path B y (f x1))
  : Path (fiber A B f y) (x0,p0) (x1,p1) = <i> (p @ i,sq1 @ i) where
rem0: Path A (g y) x0 = <i> comp (<k> A) (g (p0 @ i)) [ (i = 1) -> t x0, (i = 0) -> <k> g y ]
rem1: Path A (g y) x1 = <i> comp (<k> A) (g (p1 @ i)) [ (i = 1) -> t x1, (i = 0) -> <k> g y ]
p: Path A x0 x1 = <i> comp (<k> A) (g y) [ (i = 0) -> rem0, (i = 1) -> rem1 ]
fill0: Square A (g y)(g (f x0)) (g y) x0 (<i> g (p0 @ i)) rem0 (<i> g y) (t x0) =
  <i j> comp (<k> A) (g (p0 @ i)) [ (i = 1) -> <k> t x0 @ j /\ k,
    (i = 0) -> <k> g y,
    (j = 0) -> <k> g (p0 @ i) ]
fill1: Square A(g y)(g(f x1))(g y) x1 (<i>g (p1@i)) rem1 (<i> g y) (t x1) =
  <i j> comp (<k> A) (g (p1 @ i)) [ (i = 1) -> <k> t x1 @ j /\ k,
    (i = 0) -> <k> g y,
    (j = 0) -> <k> g (p1 @ i) ]
fill2: Square A (g y) (g y) x0 x1 (<k> g y) p rem0 rem1 =
  <i j> comp (<k> A) (g y) [ (i = 0) -> <k> rem0 @ j /\ k,
    (i = 1) -> <k> rem1 @ j /\ k,
    (j = 0) -> <k> g y ]
sq: Square A(g y)(g y)(g(f x0))(g(f x1))(<i>g y) (<i>g (f(p@i)))(<j>g(p0@j))(<j>g(p1@j)) =
  <i j> comp (<k> A) (fill2 @ i @ j) [ (i = 0) -> <k> fill0 @ j @ -k,
    (i = 1) -> <k> fill1 @ j @ -k,
    (j = 0) -> <k> g y,
    (j = 1) -> <k> t (p @ i) @ -k ]
sq1: Square B y y (f x0) (f x1) (<k>y) (<i> f (p @ i)) p0 p1 =
  <i j> comp (<k> B) (f (sq @ i @j)) [ (i = 0) -> s (p0 @ j),
    (i = 1) -> s (p1 @ j),
    (j = 1) -> s (f (p @ i)),
    (j = 0) -> s y ]

```

Iso = Equiv = (=)

Trivial Fiber = Pi

```

lem2 (B: U) (F: B -> U) (y: B) (x: F y)
  : Path (F y) (comp (<i>F (refl B y @ i)) x []) x
  = <j> comp (<i>F ((refl B y) @ j /\ i)) x [(j=1) -> <k>x]

```

```

FiberPi (B: U) (F: B -> U) (y: B)
  : Path U (fiber (total B F) B (trivial B F) y) (F y)
  = isoPath T A f g s t where
  T: U = fiber (total B F) B (trivial B F) y
  A: U = F y
  f: T -> A = encode B F y
  g: A -> T = decode B F y
  s (x: A): Path A (f (g x)) x = lem2 B F y x
  t (x: T): Path T (g (f x)) x = lem3 B F y x

```



```
natToMaybe: nat -> fix maybe = split
  zero  -> Fix nothing
  succ n -> Fix (just (natToMaybe n))
```

```
maybeToNat: fix maybe -> nat = split
  Fix m  -> go m where go: maybe (fix maybe) -> nat = split
    nothing -> zero
    just f  -> succ (maybeToNat f)
```

```
natMaybelso: (a: nat) -> Path nat (maybeToNat (natToMaybe a)) a = split
  zero  -> <i> zero
  succ n -> <i> succ (natMaybelso n @ i)
```

```
maybeNatIso : (a : fix maybe) -> Path (fix maybe) (natToMaybe (maybeToNat a)) a = split
  Fix m  -> go m where go: (a: maybe (fix maybe)) -> Path (fix maybe) (natToMaybe (maybeToNat (Fix a))) (Fix a) = split
    nothing -> <i> Fix nothing
    just f  -> <i> Fix (just (maybeNatIso f @ i))
```

```
maybenat: Path U (fix maybe) nat = isoPath (fix maybe) nat maybeToNat natToMaybe natMaybelso maybeNatIso
HeteroEqu (A B:U)(a:A)(b:B)(P:Path U A B):U = PathP P a b
```

```
> HeteroEqu (fix maybe) nat (Fix nothing) zero maybenat
```

```
> transNeg (fix maybe) (nat) maybenat (succ (succ zero))
EVAL: Fix (just (Fix (just (Fix nothing))))
> trans (fix maybe) (nat) maybenat (Fix nothing)
EVAL: zero
```

Fix Maybe = Nat

```
data maybe (A: U) = nothing | just (a: A)
data fix (F:U->U) = Fix (point: F (fix F))
```

```
data nat = zero | succ (n: nat)
```

Thank You!

<https://groupoid.space>