Intetics f(cafe) 25 July 2018 Freud House, Kyiv, Ukraine

Namdak Tonpa

HoTT: The Language of Space

Groupoid Infinity

Abstract

Cubical Base Library

Homotopy Type Theory (HoTT) is the most advanced programming language in the domain of intersection of several theories: algebraic topology, homological algebra, higher category theory, mathematical logic, and theoretical computer science. That is why it can be considered as a language of space, as it can encode any existent mathematics.

During this lecture on HoTT, we are trying to encode as much mathematics in the programming language as possible.

Talk Structure

I. Foundations

— MLTT
— Inductive Types, Induction
— IPL and Elements of Set Theory
— Control, Recursive Schemes
— Equiv, Iso, Univalence
— Higher Inductive Types
— Modalities

II. Mathematics

— Category Theory
— Topos Theory
— Basic Algebra
— Ordinals, Mahlo Universe
— Differential Topology
— FIber Bundles and Hopf Fibtations
— K-Theory

I. Foundations

λ-Calculuses
in Extended Lambda Cube

CoC: | $* \to *$ | $\square \to *$ | $* \to \square$ | $\square \to \square$ |

Fω: | $* \to *$ | $\square \to *$ | | $\square \to \square$ |

P2: | $* \to *$ | $\square \to *$ | $* \to \square$ |

AUT: | $* \to *$ | | $* \to \square$ |

F: | $* \to *$ | $\square \to *$ |

System Fω:
Haskell, Scala, 1ML
Almost CoC
No Types
On Values

Infinity Topoi
Agda, Coq, Lean, Om

System F:
ML, Miranda,
OCaml

POLYREC

CoC: Morte, Henk

STLC

Untypled SLC:
Erlang, LISP,
JavaScript

P_2:
No Types
On Types

No Terms
On Types

AUTOMATH

http://ttic.uchicago.edu/~dreyer/
course/papers/barendregt.pdf
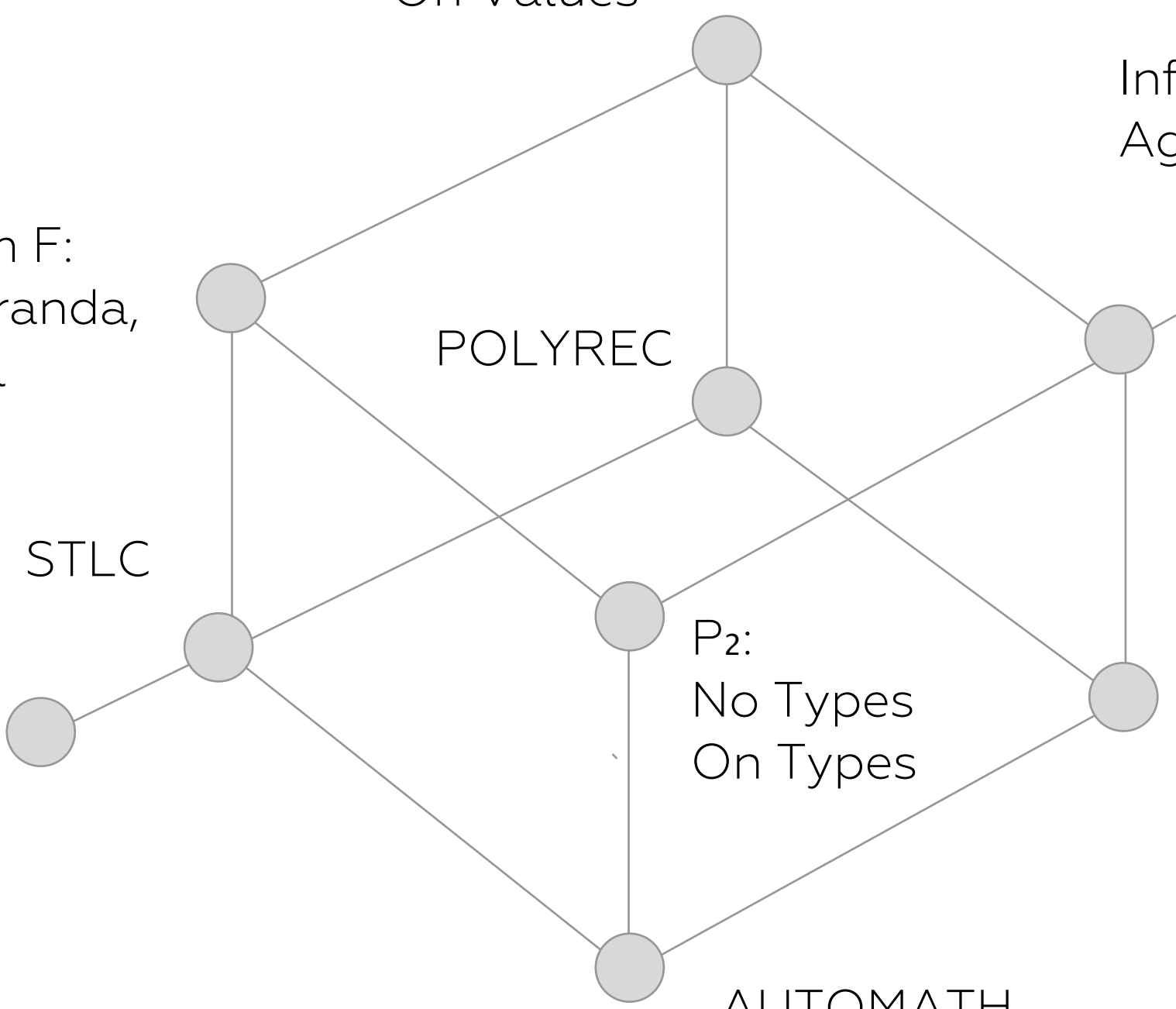
MLTT 1972

Type Theory as new Foundations of Mathematics

U : U — Single Universe Model — MLTT 1972, CoC 1988.

x : A — x is a point (Star) in space A (Box)

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

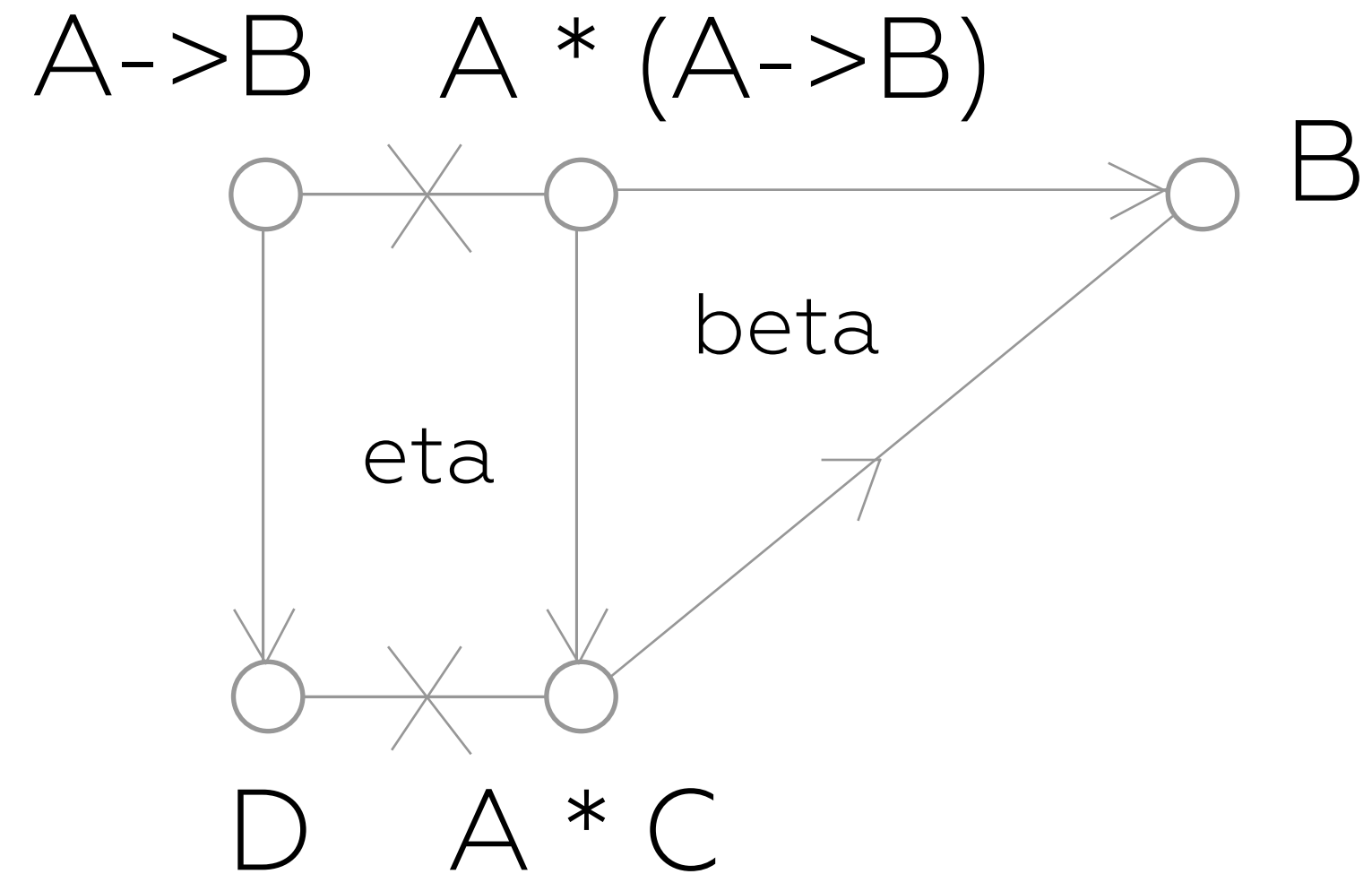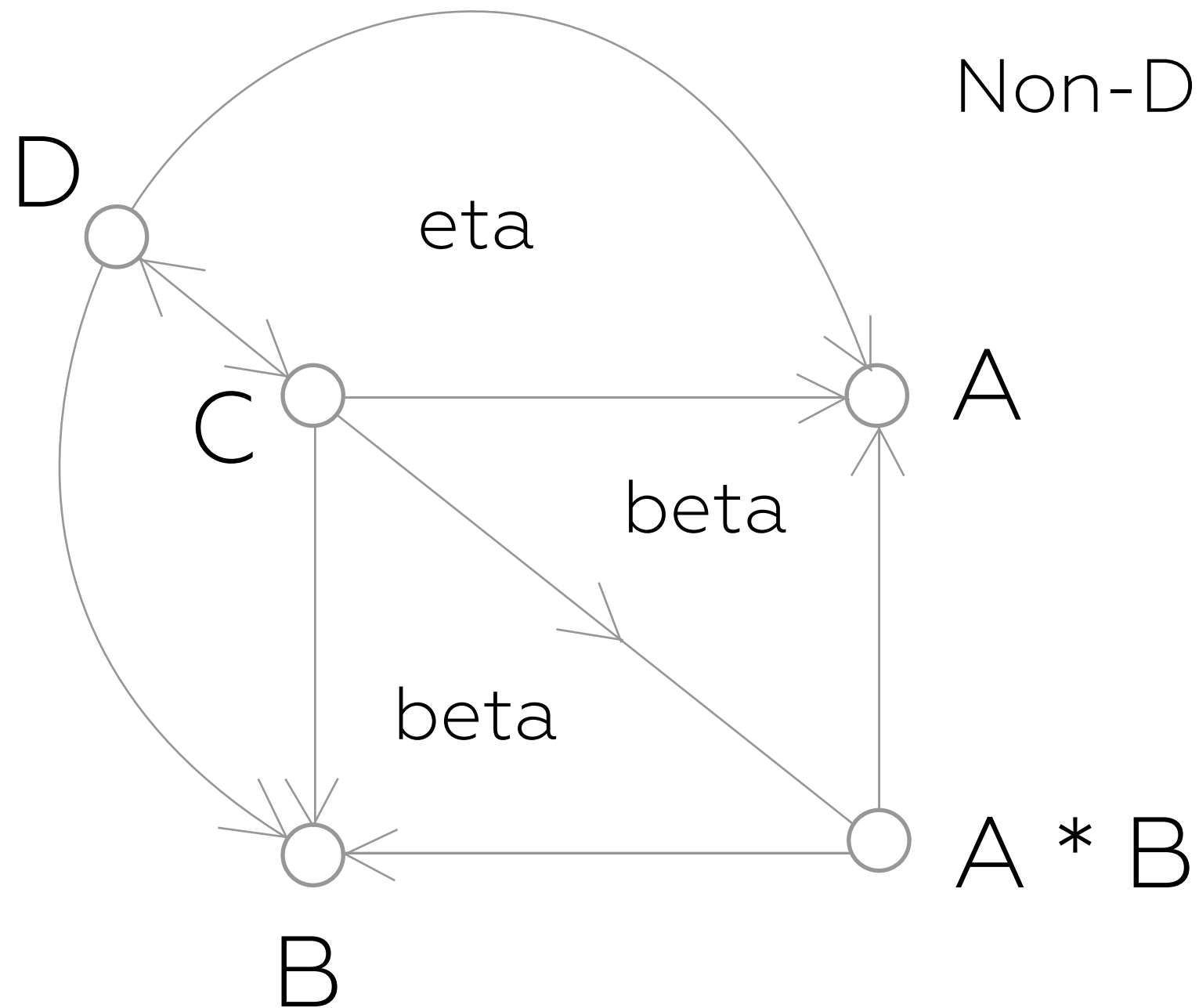|	Π	Σ
1. Formation Rules	(x:A) -> B(x)	(x:A) * B(x)
2. Introduction Rules	\ (x:A) -> B(x)	(a,b)
3. Elimination Rules	f a = B(a)	.1, .2
4. Computational Rules	two	three

Beta and Eta

Duality of Intro and Elim and its Uniqueness
Non-Dep Case (CCC). Homework: Proof LCCC case.

MLTT 1975, 1984

Grothendieck Universe (containing all sets), Countable Universes

$U_0 : U_1 : U_2 : U_3 : \dots \infty$ — infinte hierarchy of universes

$S\ (n: nat) = U_n$

$A_1\ (n\ m: nat) = U_n : U_m$ where $[m>n]$ — cumulative, $[n+1=m]$ — non-cumulative

$R_1\ (m\ n: nat) = U_m \longrightarrow U_n : U_x$ where $[x=max(m,n)]$ — predicative,

$[x=n]$ — impredicative

1. Formation	data nat	data list	x:A = y:A	data W
2. Introduction	zero, succ	nil, cons	refl A x	sup
3. Elimination	natInd	listInd	J	wInd
4. Computational	Beta, Eta	Beta, Eta	Beta, Eta	Beta, Eta

Intuitionistics Propositional Logic

According to Brouwer–Heyting–Kolmogorov interpretation

∀	∃	Path	0	1	+
x:A -> B(x)	x:A * B(x)	x:A = y:A	data empty	data unit	data either
\ (x: A) -> B(x)	(x,B(x))	refl A x		tt	inl, inr
f a = B(a)	pr1, pr2	J	elim0	elim1	eitherInd
Beta, Eta	Beta, Eta	Eta	Beta, Eta	Beta, Eta	Beta, Eta

Proto (Prelude)

For run-time and I/O applications

maybe	either	stream	bool	vector	fin
U -> U	U -> U -> U	U -> U	U	Nat -> U	Nat -> U
nothing, just	inl, inr	cons	true, false	vz, vs	fz, fs
maybeInd	eitherInd	streamInd	boolInd	vecInd	finInd
Beta, Eta	Beta, Eta	Beta, Eta	Beta, Eta	Beta, Eta	Beta, Eta

Induction Principle

natCase (C:U) (a b: C): nat -> C
  = split { zero -> a ; succ n -> b }

natRec (C:U) (z: C) (s: nat->C->C): (n:nat) -> C
  = split { zero -> z ; succ n -> s n (natRec C z s n) }

natInd (C:nat->U) (z: C zero) (s: (n:nat)->C(n)->C(succ n)): (n:nat) -> C(n)
  = split { zero -> z ; succ n -> s n (natInd C z s n) }

Induction Principle could be the ultimate programming tool.

Inductive AST
Data Definition and Case Analysis

```
data tele    (A: U) = emp | tel (n: name) (b: A) (t: tele A)
data branch (A: U) = br (n: name) (args: list name) (term: A)
data label   (A: U) = lab (n: name) (t: tele A)
data ind = data_  (n: name) (t: tele lang) (labels:     list (label lang))
          | case    (n: name)  (t: lang)        (branches: list (branch lang))
          | ctor    (n: name)                    (args:       list lang)
```

Induction Principle could be the ultimate programming tool.

Pi Type : Definition

Family of Types, Fibrations, Fiber Space A->U, Fiber B(x),
Section b(x), Space of Sections Pi(A,B)

Syntax

<> ::= #option
T ::= #identifier
U ::= * < #number >
O_1 ::= U | T | (O) | O O | O -> O
    | \ (l: O) -> O | (l: O) -> O

Model

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)

Pure Type System (PTS), Single Axiom System, Calculus of Constructions (CoC)
Henk, Morte, Om and many many others.

Pi Type : Inference Rules

Formal Definition

Pi (A: U) (P: A -> U) : U = (x:A) -> P(x)

lambda (A : U) (B: A -> U) (a : A) (b: B a): A -> B a = ?

app (A : U) (B: A -> U) (a : A) (f: A -> B a): B a = ?

Beta (A:U) (B:A->U) (a:A) (f: A->B a) : Path (B a) (app A B a (lam A B a (f a))) (f a)

Eta (A: U) (B: A -> U) (a: A) (f: A -> B a) : Path (A -> B a) f (\(x:A) -> f x)

One beta rule and one eta rule for Pi types.

Sigma Type : Definition

Total Space Sigma(A,B), Point in Base with Section (a,b)

Syntax $\quad O_2 := (x: O) * O \mid (O,O) \mid O.1 \mid O.2$

Model
```
data exists = sigma (n: name) (a b: lang)
            | pair (a b: lang)
            | fst (p: lang)
            | snd (p: lang)
```

Sigma is a part of the MLTT earliest core.
It models Type Refinement and Proofs by Existance (Construction).
Sigma is a chain link of telescopes (contexts), the curried notion of records.

Sigma Type : Inference Rules

Existential Quantifier

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 = ?
pr1 (A: U) (B: A -> U) (x: Sigma A B): A = ?
pr2 (A: U) (B: A -> U) (x: Sigma A B): B (pr1 A B x) = ?
Beta1 (B: A -> U) (a: A) (b: B a) -> Path A a (pr1 A B (pair A B a b)))
Beta2 (B: A -> U) (a: A) (b: B a) -> Path (B a) b (pr2 A B (a,b)))
Eta (B: A -> U) (p: Sigma A B) -> Path (Sigma A B) p (pr1 A B p,pr2 A B p))

sigRec (A:U)(B:A->U)(C: U) (g:(x:A)->B(x)->C) (p: Sigma A B): C = g p.1 p.2
sigInd (A:U)(B:A->U)(C:Sigma A B->U)
        (p: Sigma A B)(g:(a:A)(b:B(a))->C(a,b)):C p=g p.1 p.2

Sigma Type in Pi

Typing and Introduction Rules in Church-Bohm-Berarducci Encoding

```
-- 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 Type in Pi

Eliminators in Church-Bohm-Berarducci Encoding

```
-- 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 )
```

Control (Haskell)

Port of Haskell-style erased 2-categorical structures for flow modeling

```
pure_sig    (F:U->U):U=   (A: U) ->                   A  -> F A
appl_sig    (F:U->U):U= (A B: U) ->   F (A -> B) -> F A -> F B
fmap_sig    (F:U->U):U= (A B: U) ->     (A -> B) -> F A -> F B
bind_sig    (F:U->U):U= (A B: U) ->    F A ->(A  -> F B)-> F B
functor: U = (F: U -> U) * fmap_sig F
applicative: U = (F: U -> U) * (_: pure_sig F) * (_: fmap_sig F) * appl_sig F
monad: U = (F:U->U)*(_:pure_sig F)*(_:fmap_sig F)*(_:appl_sig F) * bind_sig F
FUNCTOR: U = (f: functor) * isFunctor f
APPLICATIVE: U = (f: applicative) * (_: isFunctor (f.1,f.2.2.1)) * isApplicative f
MONAD: U = (f: monad) * (_: isFunctor (f.1,f.2.2.1))
            * (_: isApplicative (f.1,f.2.1,f.2.2.1,f.2.2.2.1)) * isMonad f
```

F-Algebras

Inductive Type Modeling with Varmo Vene style Recursion Schemes

```
data fix (F:U->U) = Fix (point: F (fix F))
data nu (F:U->U) (A B:U) = CoBind (a: A) (f: F B)
data cofree (F:U->U) (A:U)   = CoFree  (_: fix (nu F A))
ind   (F: U -> U) (A: U): U = (in_: F (fix F) -> fix F)  * (in_rev: fix F -> F (fix F))
* ((F A -> A) -> fix F -> A) * (cofree_: (F (cofree F A) -> A) -> fix F -> A)
inductive   (F: functor) (A: U): ind F.1 A = (in_ F.1,out_ F.1,cata A F,histo A F,tt)
```

Backported to cubicaltt.

Bishop's Constructive Analysis

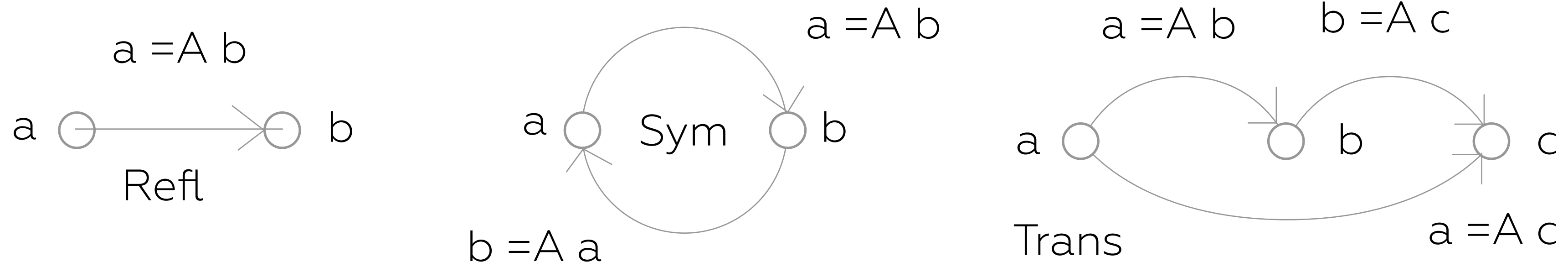Reflexivity, Transitivity, Symmetry

Setoid (A: U): U
  = (Carrier: A)
  * (Equ: (a b: A) -> Path A a b)
  * (Refl: (x: A) -> Equ x x)
  * (Trans: $(x_1,x_2,x_3$: A) -> Equ x_1 x_2 -> Equ x_2 x_3 -> Equ x_1 x_3)
  * (Sym: $(x_1,x_2$: A) -> Equ x_1 x_2 -> Equ x_2 x_1)

Globular Theory

Multidimentional Equality

a =A b

a ⭘———⭘ b
   A

a   =A   b
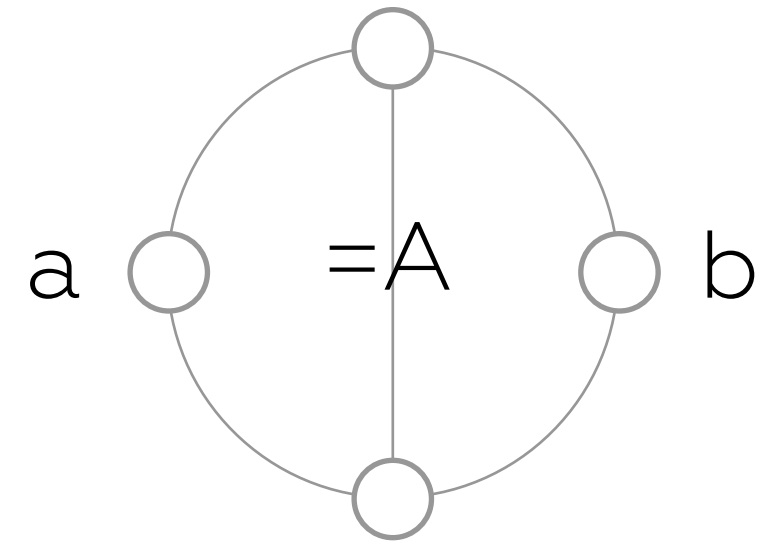
((a =A b) =(=A) (a =A b))

((a =A b) =(=A) (a =A b)) =(=(=A)) ((a =A b) =(=A) (a =A b))

a =A b

a   b

a =A b
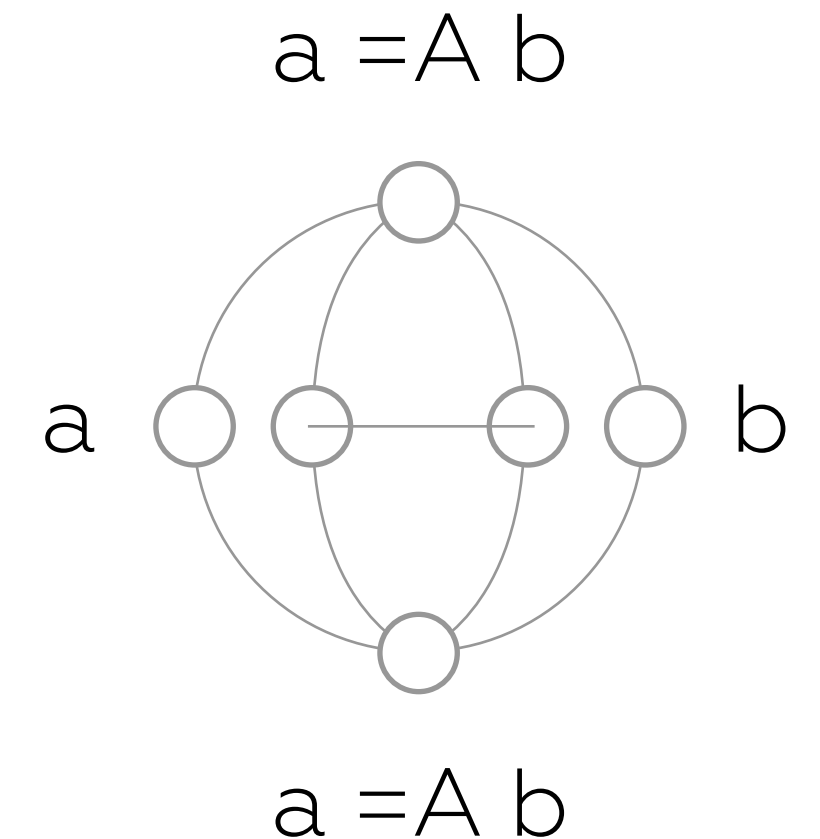
Equ Type a la Martin-Löf

Path (A: U) (a b: A): U = axiom — PathP (<i>A) a b
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)
refl   (A: U) (a: A): Equ A a a = <i> a
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
Comp   (A : U) (a : A) (C : (x : A) -> Path A a x -> U)
          (d : C a (refl A a)) : Path (C a (refl A a)) d (J A a C d a (refl A a))

Path Types as Cubes

Syntax and Model

Syntax

x : [PathP p a b, p = (i: I) -> A]

a : A ◯ ——————▶ ◯ b : A

de Morgan: 1-i | i | i ∧ j | i ∨ j

Model

```
data lang = hts | ...
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)
```

n-Types

```
Path         (A : U) : U = (a b : A) -> PathP (<i> A) a b
isContr      (A : U): U = (x: A) * ((y: A) -> Path A x y)
isProp       (A : U) : U = (a b : A) -> Path A a b
isSet        (A : U) : U = (a b : A) -> isProp      (Path A a b)
isGroupoid   (A : U) : U = (a b : A) -> isSet       (Path A a b)
isGr_2       (A : U) : U = (a b : A) -> isGroupoid (Path A a b)
isGr_3       (A : U) : U = (a b : A) -> isGr_2      (Path A a b)

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
```

Subtyping in MLTT
Subsets and Subtypes

```
hsubtypes  (X: U): U = X -> PROP
subset (A: U) (_: isSet A): U = A -> PROP


sethsubtypes (X : U) : isSet (hsubtypes X)
hsubtypespair (A B: U) (H0: hsubtypes A) (H1: hsubtypes B) (x: prod A B): PROP


subtypeEquality (A: U) (B: A -> U)
                (pB: (x : A) -> isProp (B x))
                (s t: Sigma A B) : Path A s.1 t.1 -> Path (Sigma A B) s t


iseqclass (X : U) (R : hrel X) (A : hsubtypes X) : U
propiseqclass (X : U) (R : hrel X) (A : hsubtypes X) : isProp (iseqclass X R A)
```

Elements of Set Theory

ac   (A B: U) (R: A -> B -> U): (p: (x:A)->(y:B)*(R x y)) -> (f:A->B)*((x:A)->R(x)(f x))
      = \(g: (x:A)->(y:B)*(R x y)) -> (\(i:A)->(g i).1,\(j:A)->(g j).2)

total (A:U) (B C : A->U) (f : (x:A) -> B x -> C x) (w:Sigma A B) : Sigma A C
      = (w.1,f (w.1) (w.2))

Prop Logic

```
efq        (A: U): empty -> A = emptyRec A
neg        (A: U): U = A -> empty
dneg       (A:U) (a:A): neg (neg A) = \(h: neg A) -> h a
neg        (A: U): U = A -> empty
dec        (A: U): U = either A (neg A)
stable     (A: U): U = neg (neg A) -> A
discrete   (A: U): U = (a b: A) -> dec (Path A a b)


propDec (A : U) (h : isProp A) : isProp (dec A)
propAnd (A B : U) (pA : isProp A) (pB : isProp B) : isProp (prod A B)
propNeg (A:U) : isProp (neg A)
propN0 : isProp empty
```

Homotopy

```
data I = i0
       | i1
       | seg <i> [(i=0) -> i0, (i=1) -> i1]

pathToHtpy (A: U) (x y: A) (p: Path A x y): I -> A
  = split { i0 -> x; i1 -> y; seg @ i -> p @ i }

homotopy  (X Y: U) (f g: X -> Y)
          (p: (x: X) -> Path Y (f x) (g x))
          (x: X): I -> Y = pathToHtpy Y (f x) (g x) (p x)
```
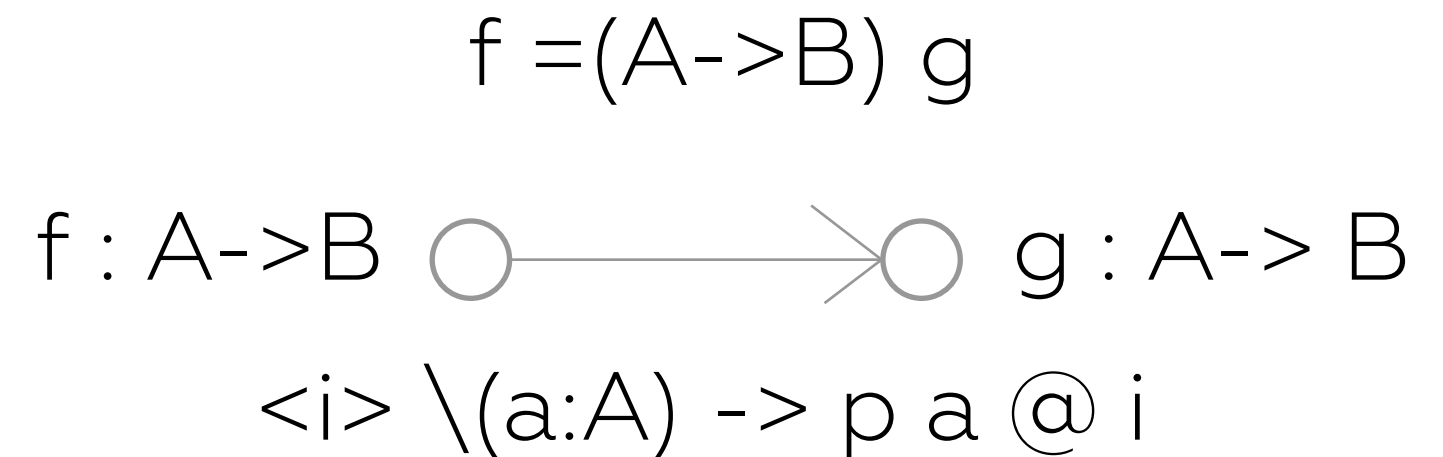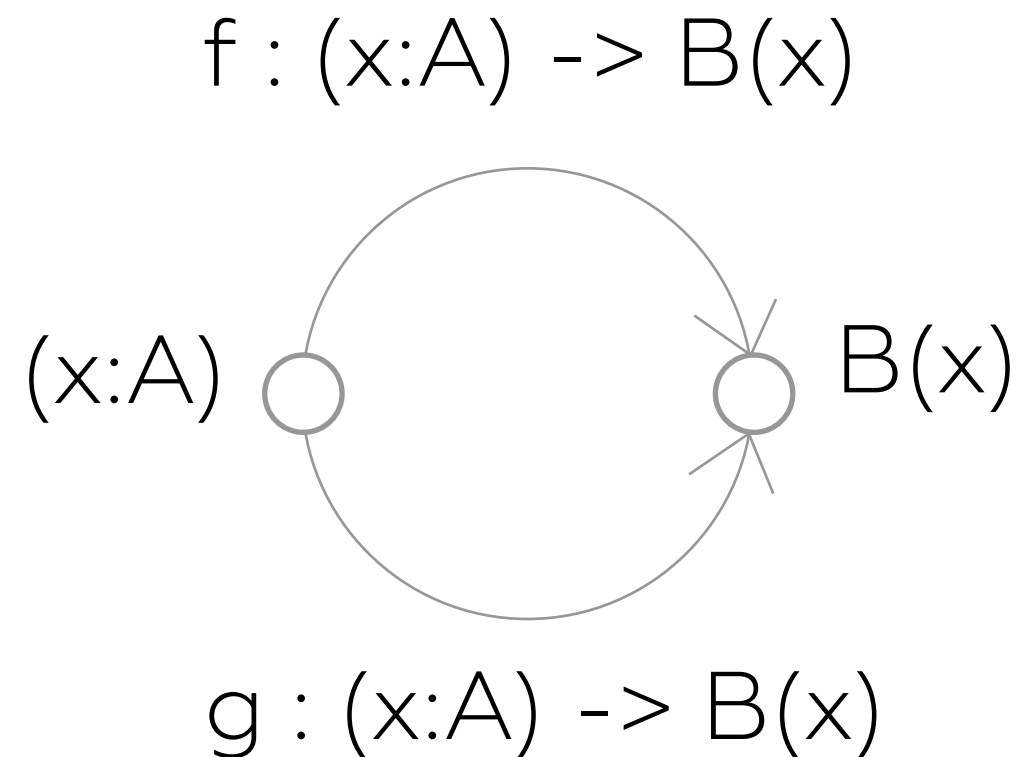
FunExt

```
funext  (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
    = <j> (\(x : A) -> homotopy A B f g p x (seg{l} @ j))
```

f : (x:A) -> B(x)

(x:A) ◯　　　◯ B(x)

g : (x:A) -> B(x)

f =(A->B) g

f : A->B ◯ ———————> ◯ g : A-> B

<i> \(a:A) -> p a @ i

FunExt

Formation, Intro, Elim, Beta, Eta

funext_form (A B: U) (f g: A -> B): U = Path (A -> B) f g

funext (A B: U) (f g: A -> B) (p: (x:A) -> Path B (f x) (g x)) : funext_form A B f g

happly (A B: U) (f g: A -> B) (p: funext_form A B f g) (x: A) : Path B (f x) (g x)

funext_Beta (A B: U) (f g: A -> B) (p: (x:A) -> Path B (f x) (g x))
            : (x:A) -> Path B (f x) (g x)

funext_Eta (A B: U) (f g: A -> B) (p: funext_form A B f g)
           : Path (funext_form A B f g) (funext A B f g (happly A B f g p)) p

Weak Equivalence

Fibrational

fiber (A B: U) (f: A -> B) (y: B): U = (x: A) * Path B y (f x)
isEquiv (A B: U) (f: A -> B): U = (y: B) -> isContr (fiber A B f y)
equiv (A B: U): U = (f: A -> B) * isEquiv A B f
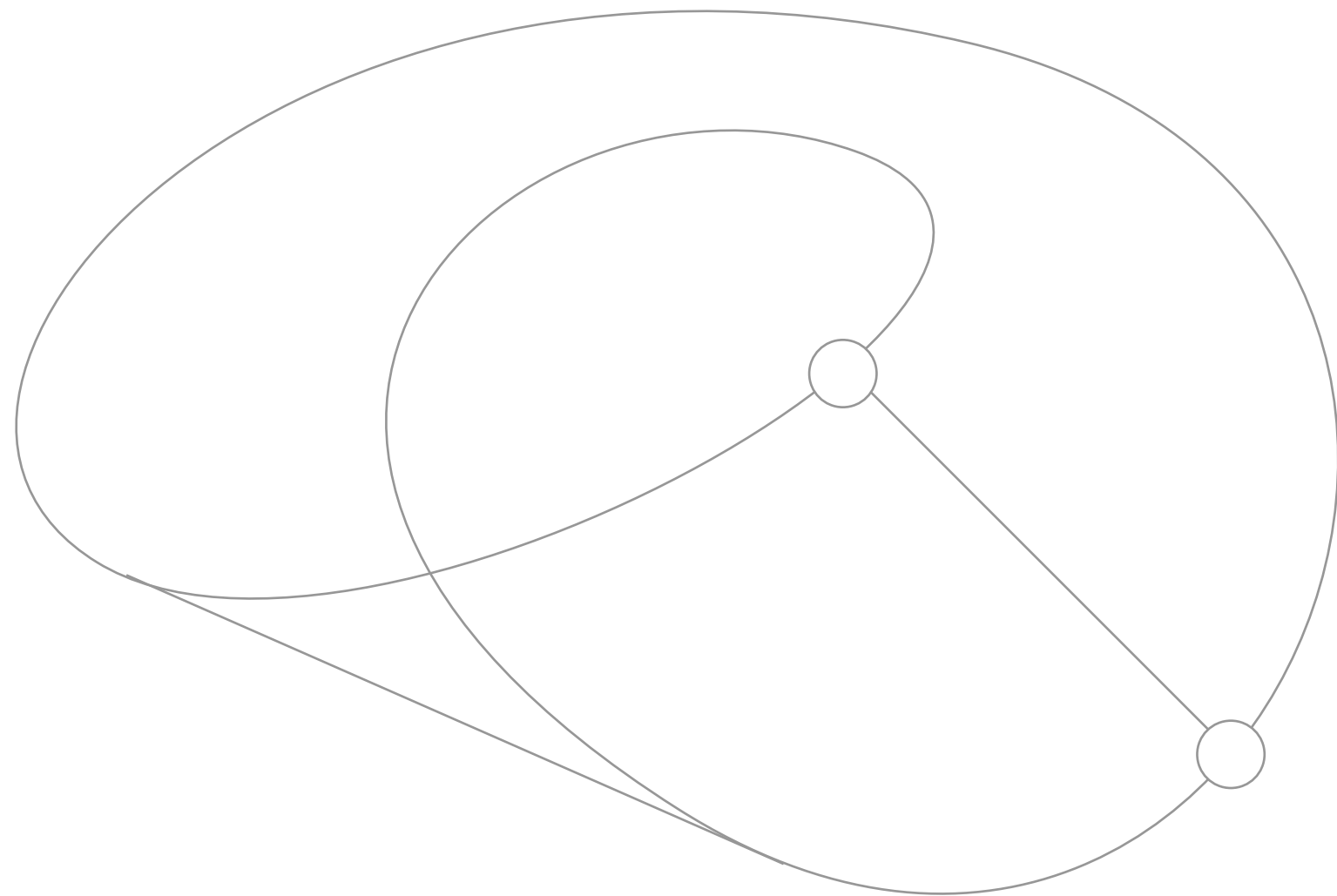
Fiber Bundle: F -> E -> B
Moebius E = S^1 'twisted *' [0,1]
Trivial: E = B * F
p : total -> B
F = fiber : B -> total
total = (y: B) * fiber(y)

Fiber=Pi (B: U) (F: B -> U) (y: B)
  : Path U (fiber (total B F) B (trivial B F) y) (F y)

Isomorphism

```
isIso (A B: U): U        --- A = XML, B = JSON
  = (f: A -> B)
  * (g: B -> A)
  * (s: section A B f g)            isoPath (A B: U) (f: A -> B) (g: B -> A)
  * (t: retract A B f g)                (s: section A B f g) (t: retract A B f g): Path U A B
  * unit                          = <i> Glue B [ (i = 0) -> (A,f,isoToEquiv A B f g s t),
                                                 (i = 1) -> (B,idfun B,idIsEquiv B) ]
iso: U
  = (A: U)                        isoToPath (i: iso): Path U i.1 i.2.1
  * (B: U)                          = 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
  * isIso A B


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

```
ua (A B: U): U = equiv A B -> Path U A B
uaIntro (A B: U): ua A B
uaElim (A B: U) (p: Path U A B): equiv A B

uaComp (A B : U) (p : Path U A B)
  : Path (Path U A B) (uaIntro A B (uaElim A B p)) p

uaUniqueness (A B : U) (w : equiv A B)
  : Path (A -> B) w.1 (uaElim A B (uaIntro A B w)).1
```

Univalence Axiom
All Equalities Should Be Equal

Path

Path(Path,Equiv)

Iso(Path,Equiv)

Equiv(Path,Equiv)

Iso

Equiv
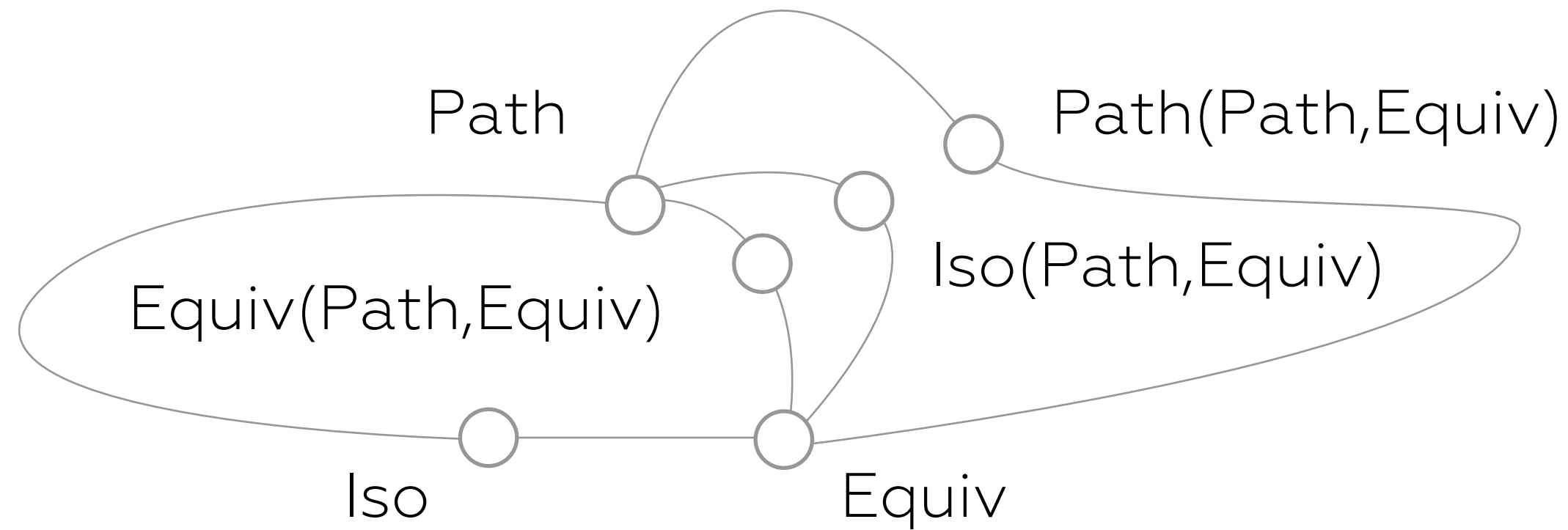
Trivial Fiber = Pi

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]

lem3 (B: U) (F: B -> U) (y: B) (x: fiber (total B F) B (trivial B F) y)
  : Path (fiber (total B F) B (trivial B F) y) ((y,encode B F y x),refl B y) x
  = <i> ((x.2 @ -i,comp (<j> F (x.2 @ -i /\ j)) x.1.2 [(i=1) -> <_> x.1.2]),<j> x.2 @ -i \/ j)

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

I. Mathematics

```
cat: U = (A: U) * (A -> A -> U)

isPrecategory (C: cat): U
  = (id: (x: C.1) -> C.2 x x)
  * (c: (x y z:C.1) -> C.2 x y -> C.2 y z -> C.2 x z)
  * (homSet: (x y: C.1) -> isSet (C.2 x y))
  * (left: (x y: C.1) -> (f: C.2 x y) -> Path (C.2 x y) (c x x y (id x) f) f)
  * (right: (x y: C.1) -> (f: C.2 x y) -> Path (C.2 x y) (c x y y f (id y)) f)
  * ((x y z w: C.1) -> (f: C.2 x y) -> (g: C.2 y z) -> (h: C.2 z w) ->
    Path (C.2 x w) (c x z w (c x y z f g) h) (c x y w f (c y z w g h)))

precategory: U = (C: cat) * isPrecategory C


Instances:
Set, Functions, Category, Functors, Commutative Monoids, Abelian Groups
```

catfunctor (A B: precategory): U
  = (ob: carrier A -> carrier B)
  * (mor: (x y: carrier A) -> hom A x y -> hom B (ob x) (ob y))
  * (id: (x: carrier A) -> Path (hom B (ob x) (ob x)) (mor x x (path A x)) (path B (ob x)))
  * ((x y z: carrier A) -> (f: hom A x y) -> (g: hom A y z) ->
    Path (hom B (ob x) (ob z)) (mor x z (compose A x y z f g))
      (compose B (ob x) (ob y) (ob z) (mor x y f) (mor y z g)))

Category Equivalence, Id and Composition Functors, Slice and Coslice

Category of Sets

Formal Model of Set Theory

Set: precategory = ((Ob,Hom),id,c,HomSet,L,R,Q) where
   Ob: U = SET
   Hom (A B: Ob): U = A.1 -> B.1
   id (A: Ob): Hom A A = idfun A.1
   c (A B C: Ob) (f: Hom A B) (g: Hom B C): Hom A C = o A.1 B.1 C.1 g f
   HomSet (A B: Ob): isSet (Hom A B) = setFun A.1 B.1 B.2
   L (A B: Ob) (f: Hom A B): Path (Hom A B) (c A A B (id A) f) f = refl (Hom A B) f
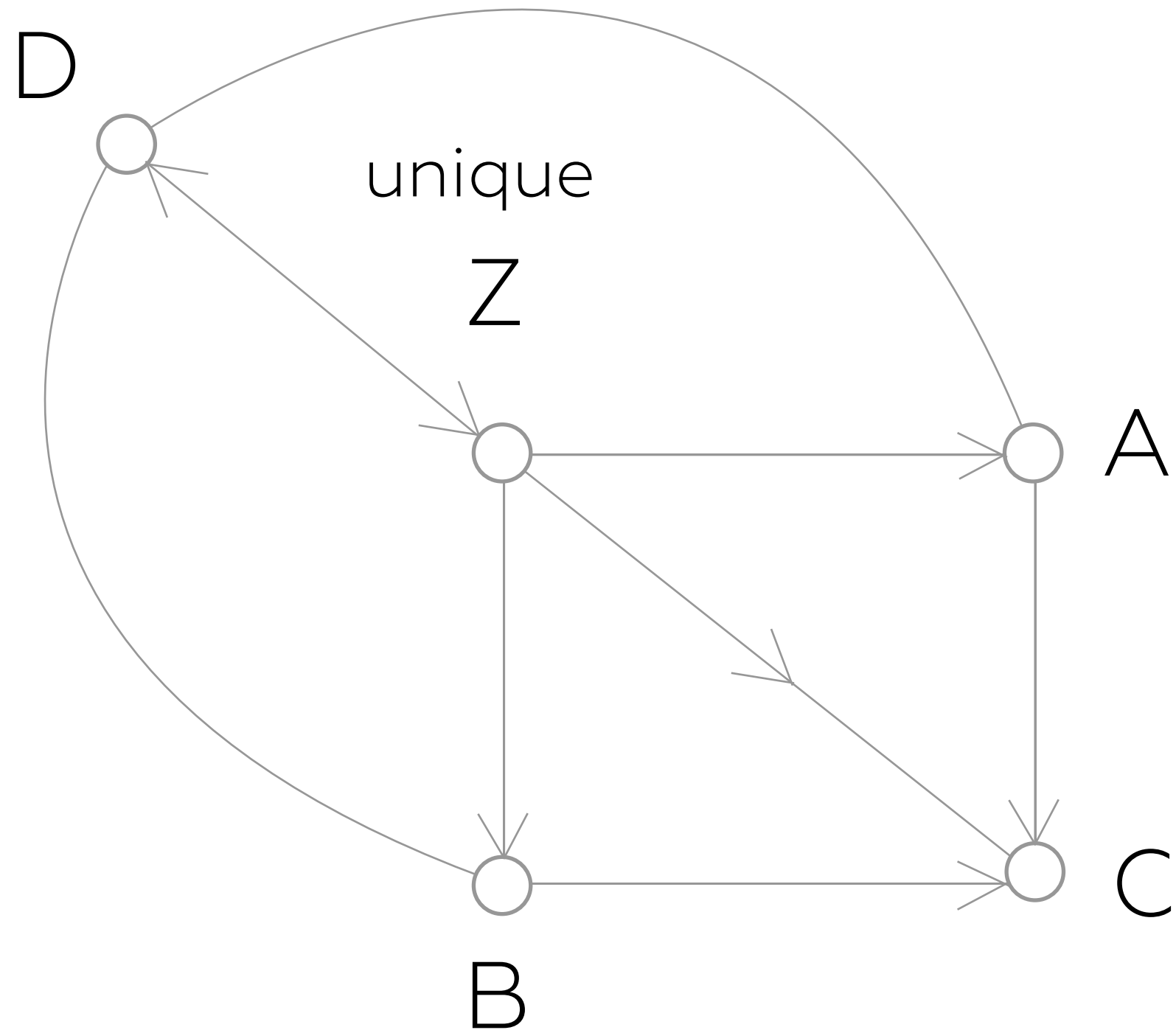   R (A B: Ob) (f: Hom A B): Path (Hom A B) (c A B B f (id B)) f = refl (Hom A B) f
   Q (A B C D: Ob) (f: Hom A B) (g: Hom B C) (h: Hom C D)
    : Path (Hom A D) (c A C D (c A B C f g) h) (c A B D f (c B C D g h))
    = refl (Hom A D) (c A B D f (c B C D g h))

Pullback Completeness
Pullbacks and Fibers as edge case


unique

Z

D

A

B

C

Examples:
Products, Fibers

Dual Examples (Pushout):
Coproducts, Cofibers

Topos Theory

Categories

subobjectClassifier (C: precategory): U
  = (omega: carrier C)
  * (end: terminal C)
  * (trueHom: hom C end.1 omega)
  * (xi: (V X: carrier C) (j: hom C V X) -> hom C X omega)
  * (square: (V X: carrier C) (j: hom C V X) -> mono C V X j
      -> hasPullback C (omega,(end.1,trueHom),(X,xi V X j)))
  * ((V X: carrier C) (j: hom C V X) (k: hom C X omega)
      -> mono C V X j
      -> hasPullback C (omega,(end.1,trueHom),(X,k))
      -> Path (hom C X omega) (xi V X j) k)

Topos (cat: precategory): U
  = (rezk: isCategory cat)
  * (cartesianClosed: isCCC cat)
  * subobjectClassifier cat
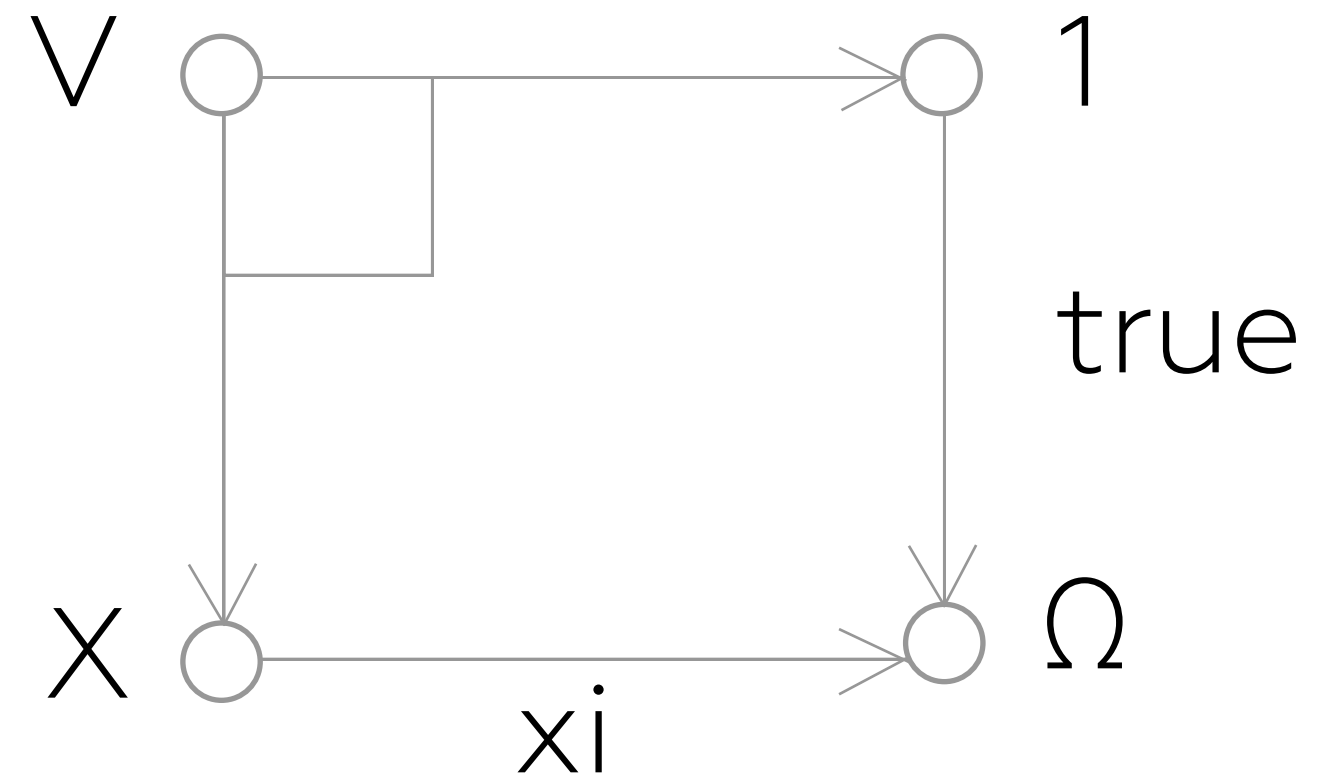
Basic Abstract Algebra

```
isMonoid (M: SET): U
  = (op: M.1 -> M.1 -> M.1)
  * (_: isAssociative M.1 op)
  * (id: M.1)
  * (hasIdentity M.1 op id)


isCMonoid (M: SET): U
  = (m: isMonoid M)
  * (isCommutative M.1 m.1)


isGroup (G: SET): U
  = (m: isMonoid G)
  * (inv: G.1 -> G.1)
  * (hasInverse G.1 m.1 m.2.2.1 inv)
```

```
isAbGroup (G: SET): U
  = (g: isGroup G)
  * (isCommutative G.1 g.1.1)


isRing (R: SET): U
  = (mul: isMonoid R)
  * (add: isAbGroup R)
  * (isDistributive R.1 add.1.1.1 mul.1)


isAbRing (R: SET): U
  = (mul: isCMonoid R)
  * (add: isAbGroup R)
  * (isDistributive R.1 add.1.1.1 mul.1.1)
```

Basic Abstract Algebra

Objects and Morphisms for Categorical Setup

monoid: U = (X: SET) * isMonoid X
cmonoid: U = (X: SET) * isCMonoid X
group: U = (X: SET) * isGroup X
abgroup: U = (X: SET) * isAbGroup X
ring: U = (X: SET) * isRing X
abring: U = (X: SET) * isAbRing X

monoidhom (a b: monoid): U
  = (f: a.1.1 -> b.1.1)
  * (ismonoidhom a b f)

cmonoidhom (a b: cmonoid): U = monoidhom (a.1, a.2.1)   (b.1, b.2.1)
grouphom   (a b: group):   U = monoidhom (a.1, a.2.1)   (b.1, b.2.1)
abgrouphom   (a b: abgroup): U = monoidhom (a.1, a.2.1.1) (b.1, b.2.1.1)
cmonabgrouphom (a: cmonoid) (b: abgroup): U = monoidhom (a.1, a.2.1) (b.1, b.2.1.1)

Ordinals

Structures

```
data ord  = zero
          | succ (n:       ord)
          | lim  (f: nat  -> ord)

data ord2 = zero
          | succ (n:       ord2)
          | lim  (f: nat  -> ord2)
          | lim2 (f: ord  -> ord2)

data ord3 = zero
          | succ (n:       ord3)
          | lim  (f: nat  -> ord3)
          | lim2 (f: ord  -> ord3)
          | lim3 (f: ord2 -> ord3)
```

http://www.cse.chalmers.se/~coquand/ordinal.ps

Mahlo Universe

Structures

```
data V
    = pi_  (x: V) (y: Elv x -> V)
    | uni_ (f: (x: V) -> (Elv x -> V) -> V)
        (g: (x: V) -> (y: Elv x -> V) -> (Elv (f x y) -> V) -> V)
Elv: V -> U = split
    pi_ a b -> (x: Elv a) -> Elv (b x)
    uni_ f g -> Universe f g
```

http://www.cs.swan.ac.uk/
~csetzer/articles/uppermahlo.ps

cubical: Resolver.hs:(293,26)-
(316,29): Non-exhaustive patterns
in case

```
data Universe
    (f: (x: V) -> (Elv x -> V) -> V)
    (g: (x: V) -> (y: Elv x -> V) -> (Elv (f x y) -> V) -> V)
    = fun_ (x: Universe f g) (_: Elt f g x -> Universe f g)
    | f_   (x: Universe f g) (_: Elt f g x -> Universe f g)
    | g_   (x: Universe f g)
        (y: Elt f g x -> Universe f g)
        (z: Elv (f (Elt f g x) (\(a: Elt f g x) -> y a)))
Elt: (f: (x: V) -> (Elv x -> V) -> V) ->
    (g: (x: V) -> (y: Elv x -> V) -> (Elv (f x y) -> V) -> V) ->
    Universe f g -> V = undefined
```
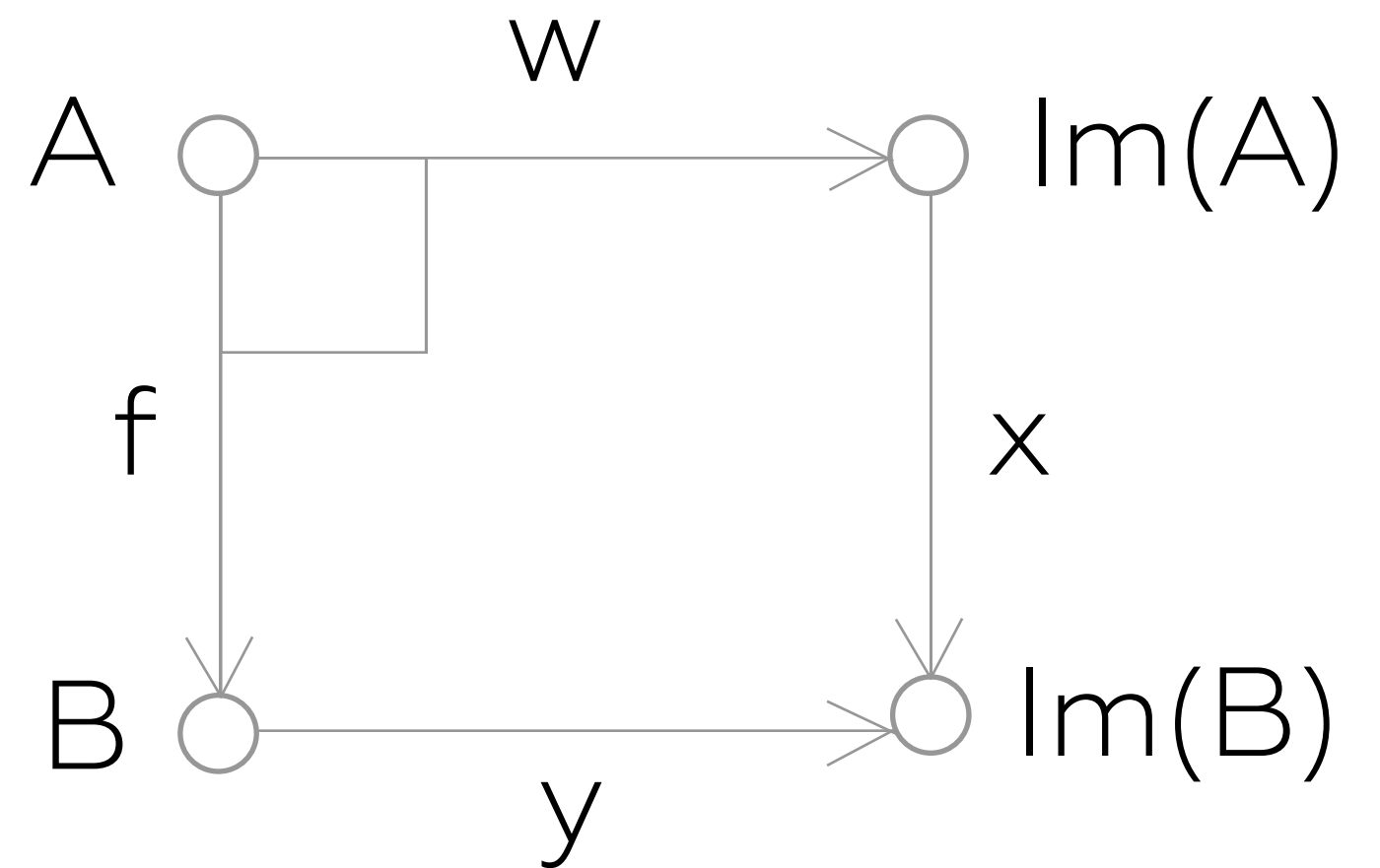
Differential Topology

```
EtaleMap (A B: U): U
  = (f: A -> B)
  * isÉtaleMap A B f


isÉtaleMap (A B: U) (f: A -> B): U
  = isPullbackSq A iA B (Im B) x y w f h where
  iA : U = Im A
  iB : U = Im B
  x: iA -> iB = ImApp  A B f
  y:  B -> iB = ImUnit B
  w:  A -> iA = ImUnit A
  c1: A -> iB = o A iA iB x w
  c2: A -> iB = o A B  iB y f
  T2: U = (a:A) -> Path iB (c1 a) (c2 a)
  h: T2 = \(a : A) -> <i> ImNaturality A B f a @ -i
```

HomogeneousStructure (V: U): U
et (A B: U): EtaleMap A B -> (A -> B)
isSurjective (A B: U) (f: A -> B): U

manifold (V': U) (V: HomogeneousStructure V'): U
  = (M: U)
 * (W: U)
 * (w: EtaleMap W M)
 * (covers: isSurjective W M (et W M w))
 * (EtaleMap W V')

https://ncatlab.org/schreiber/show/thesis+Wellen

Infinitesimal Modality

in Cohesive Topos

Im : U -> U = undefined
ImUnit (A: U) : A -> Im A = undefined
isCoreduced (A:U): U = isEquiv A (Im A) (ImUnit A)
ImCoreduced (A:U): isCoreduced (Im A)
ImApp (A B: U) (f: A -> B): Im A -> Im B
  = ImRecursion A (Im B) (ImCoreduced B) (o A B (Im B) (ImUnit B) f)
ImNaturality (A B:U) (f:A->B): (a:A)->Path (Im B)((ImUnit B)(f a))((ImApp A B f)(ImUnit A a))

ImInduction (A:U)(B:Im A->U)(x: (a: Im A)->isCoreduced(B a))
            (y:(a: A)->B(ImUnit A a)):(a:Im A)->B a

ImComputeInduction (A:U)(B:Im A->U) (c:(a:Im A)->isCoreduced(B a))
                   (f:(a:A)->B(ImUnit A a))(a:A)
                 : Path (B (ImUnit A a)) (f a) ((ImInduction A B c f) (ImUnit A a))

Higher Spheres

Fiber Bundle of Spheres

```
data S1 = base
        | loop <i> [ (i=0) -> base, (i=1) -> base]


data susp (A : U) = north
                  | south
                  | merid (a : A) <i> [ (i=0) -> north, (i=1) -> south ]
S2 : U = susp S1
S3 : U = susp S2
S4 : U = susp S3


S: nat -> U = split
  zero -> susp bool
  succ x -> susp (S x)
```

Hopf Fibrations
Fiber Bundle of Spheres

ua (A B : U) (e : equiv A B) : Path U A B = <i> Glue B [(i = 0) -> (A,e), (i = 1) -> (B,idEquiv B)]
rot: (x : S1) -> Path S1 x x = split { base -> loop1 ; loop @ i -> constSquare S1 base loop1 @ i }

mu : S1 -> equiv S1 S1 = split
  base -> idEquiv S1
  loop @ i -> equivPath S1 S1 (idEquiv S1) (idEquiv S1) (<j> \(x : S1) -> rot x @ j) @ i

H : S2 -> U = split { north -> S1 ; south -> S1 ; merid x @ i -> ua S1 S1 (mu x) @ i }
TH : U = (c : S2) * H c

Sequences

```
data Seq (A: U) (B: A -> A -> U) (X Y: A)
     = seqNil (_: A)
      | seqCons (X Y Z: A) (_: B X Y) (_: Seq A B Y Z)

fiberSeq: pointed -> pointed -> U = Seq pointed pmap
fiberNil (X: pointed): fiberSeq X X = seqNil X
fiberCons (X Y Z: pointed) (h: pmap X Y) (t: fiberSeq Y Z): fiberSeq X Z = seqCons X Y Z h t

homSeq: group -> group -> U = Seq group grouphom
homNil (X: group): homSeq X X = seqNil X
homCons (X Y Z: group) (h: grouphom X Y) (t: homSeq Y Z): homSeq X Z = seqCons X Y Z h t

abSeq: abgroup -> abgroup -> U = Seq abgroup abgrouphom
abNil (X: abgroup): abSeq X X = seqNil X
abCons (X Y Z: abgroup) (h: abgrouphom X Y) (t: abSeq Y Z): abSeq X Z = seqCons X Y Z h t
```

Chain Complexes

ChainComplex: U
 = (head: abgroup)
 * (chain: nat -> abgroup)
 * (augment: abgrouphom (chain zero) head)
 * ((n: nat) -> abgrouphom (chain (succ n)) (chain n))

CochainComplex: U
 = (head: abgroup)
 * (cochain: nat -> abgroup)
 * (augment: abgrouphom head (cochain zero))
 * ((n: nat) -> abgrouphom (cochain n) (cochain (succ n)))

https://github.com/groupoid/cafe

Thank You!

https://groupoid.space