

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

Namdak Tonpa

HoTT: The Language of Space

Groupoid Infinity

The HoTT Lineage

- Nicolaas Govert de Bruijn [AUTOMATH]
- Per Martin-Löf, Henk Barendregt [MLTT, λ -Cube]
- Robin Milner [ML, LCF, π -Calculus]
- Gérard Pierre Huet [CAML]
- Christine Paulin-Mohring [CiC]
- Peter Dybjer, Martin Hofmann, R.A.G. Seely [CwF, CwA, LCCC]
- Thierry Coquand [CoC, Coq, CCHM]
- Robert William "Bob" Harper, Jr. [HoTT, RedPRL, ML, Twelf]
- Steve Awodey, Vladimir Voevodsky [HoTT, C-Systems]
- Cyril Cohen, Simon Huber, Anders Mörtberg [CCHM, cubicaltt, yacctt]

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

Slightly based on HoTT Chapters

I. Foundations

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

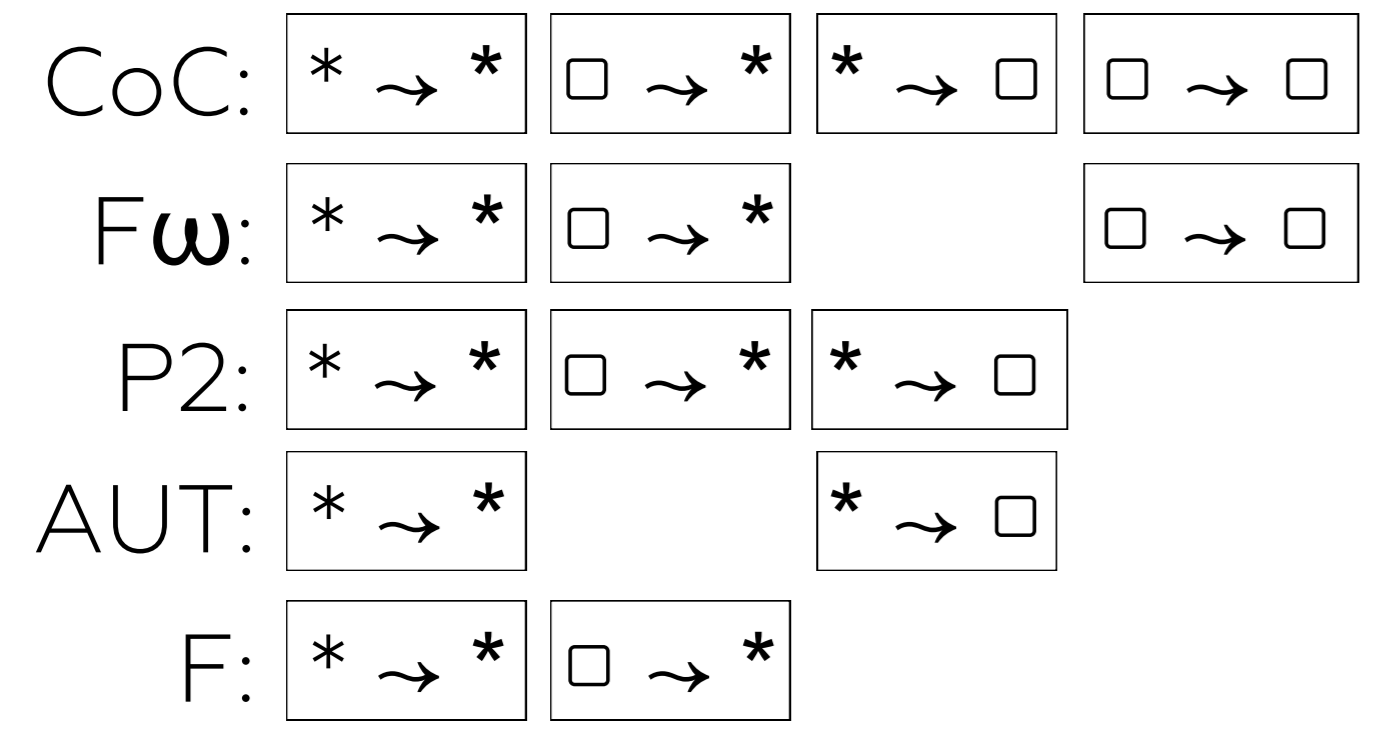
II. Mathematics

- Category Theory, Topos Theory
- Basic Algebra
- Ordinals, Mahlo Universe
- Differential Topology
- Fiber Bundles and Hopf Fibrations
- K-Theory
- Sequences, Chain Complexes

I. Foundations

λ -Calculi

in Extended Lambda Cube



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

System F:
 ML, Miranda,
 OCaml

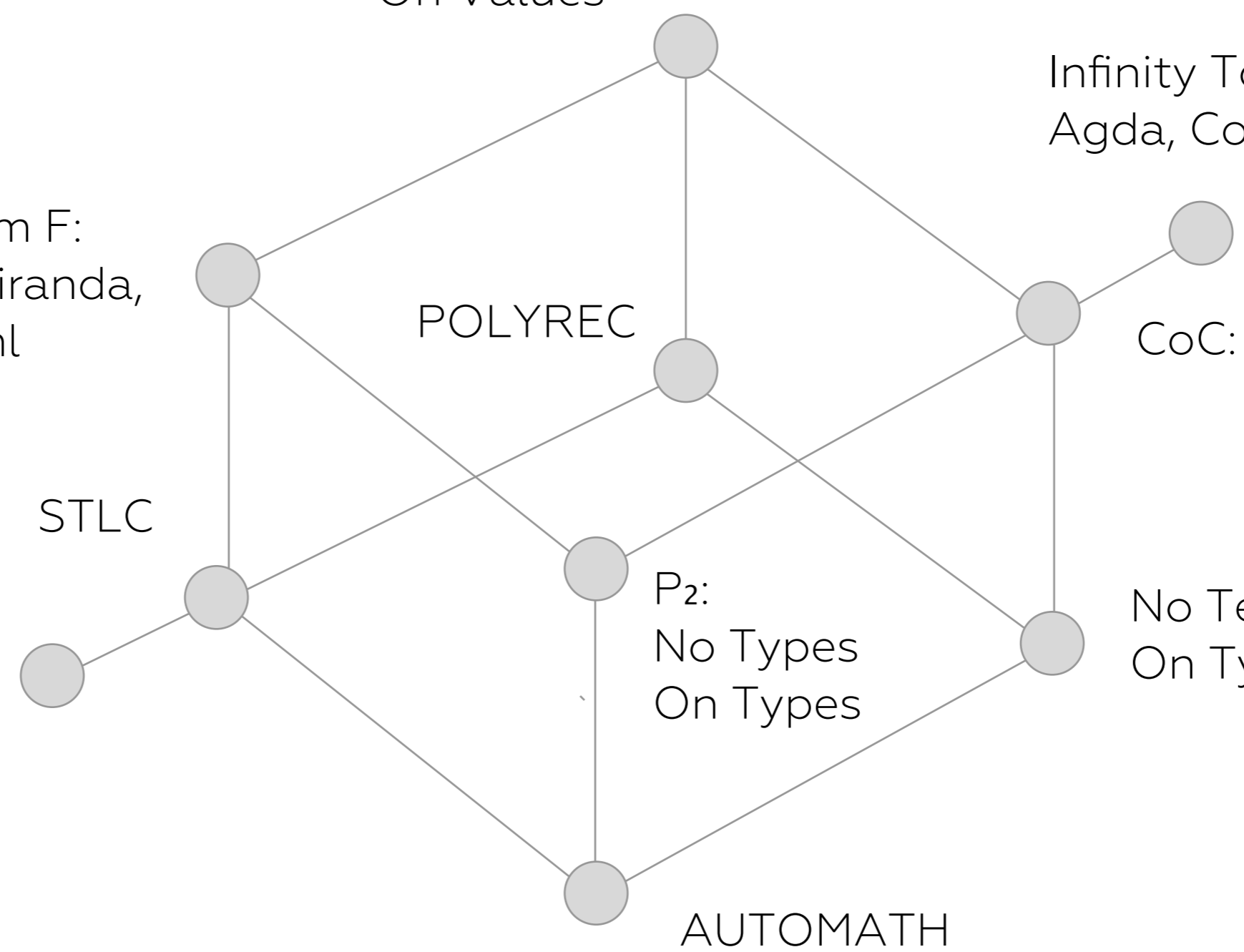
Infinity Topoi
 Agda, Coq, Lean, Om

CoC: Morte, Henk

Untyped SLC:
 Erlang, LISP,
 JavaScript

P₂:
 No Types
 On Types

No Terms
 On Types



OMPTS

Type Inference Algorithm (Pure Case Analysis) in Erlang

```
type (:star, N) D → (:star, N+1)
  (:var, N, I) D → :true = proplists:defined N B, om:keyget N D I
(:pi, N, O, I, O) D → (:star ,h (star (type I D)) (star (type O [(N,norm I)|D])))
(:fn, N, O, I, O) D → let star (type I D), Ni = norm I in (:pi,N,O,Ni,type(O,[(N,Ni)|D]))
  (:app, F, A) D → let T = type(F,D), (:pi,N,O,I,O) = T, :true = eq I (type A D)
                    in norm (subst O N A)
```

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 definitionally equal objects of type A

1. Formation Rule

2. Introduction Rule(s)

3. Elimination Rule(s)

4. Computational Rule(s)

Π

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

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

$f a$

two

Σ

$(x:A) * B(x)$

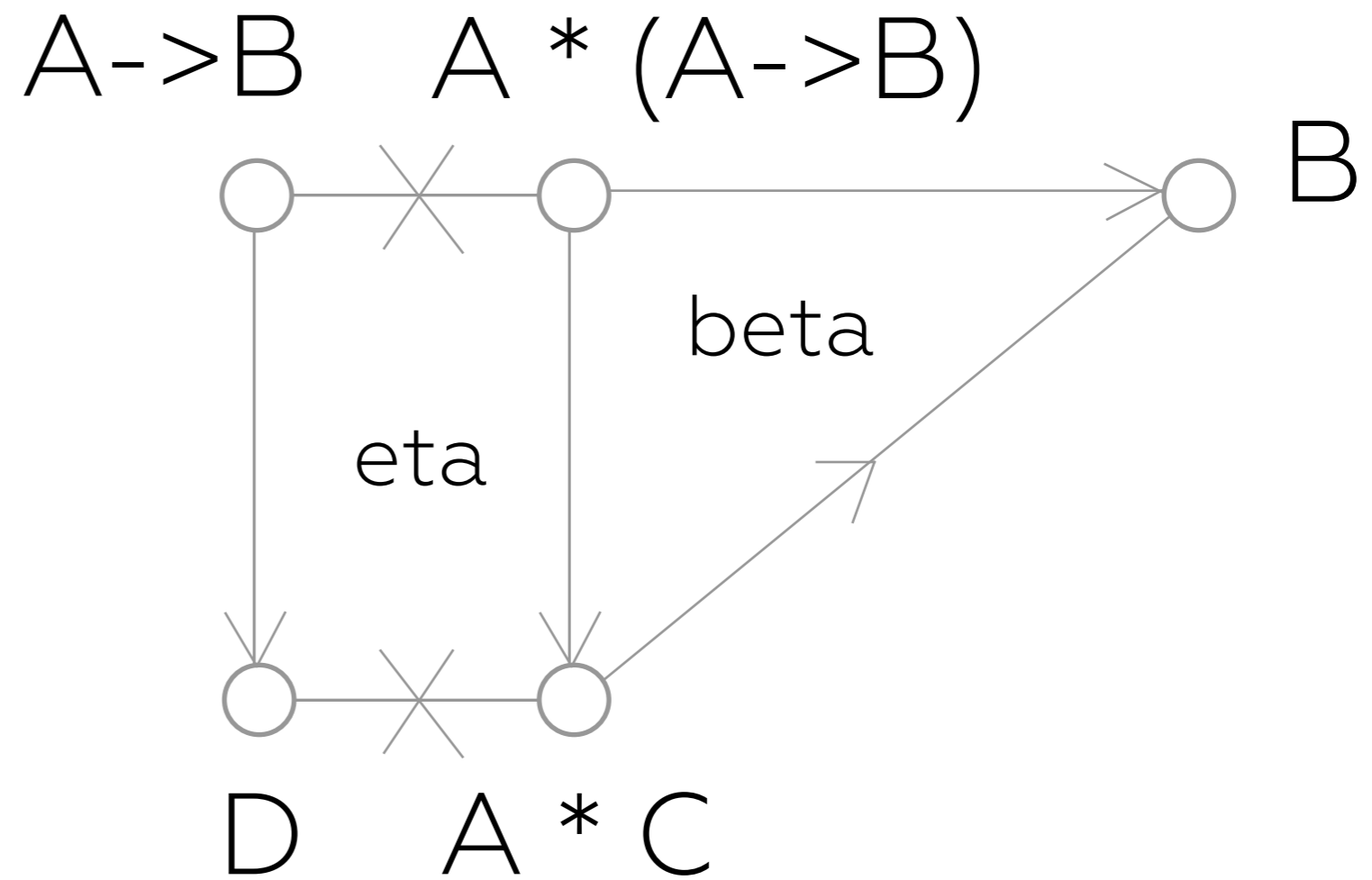
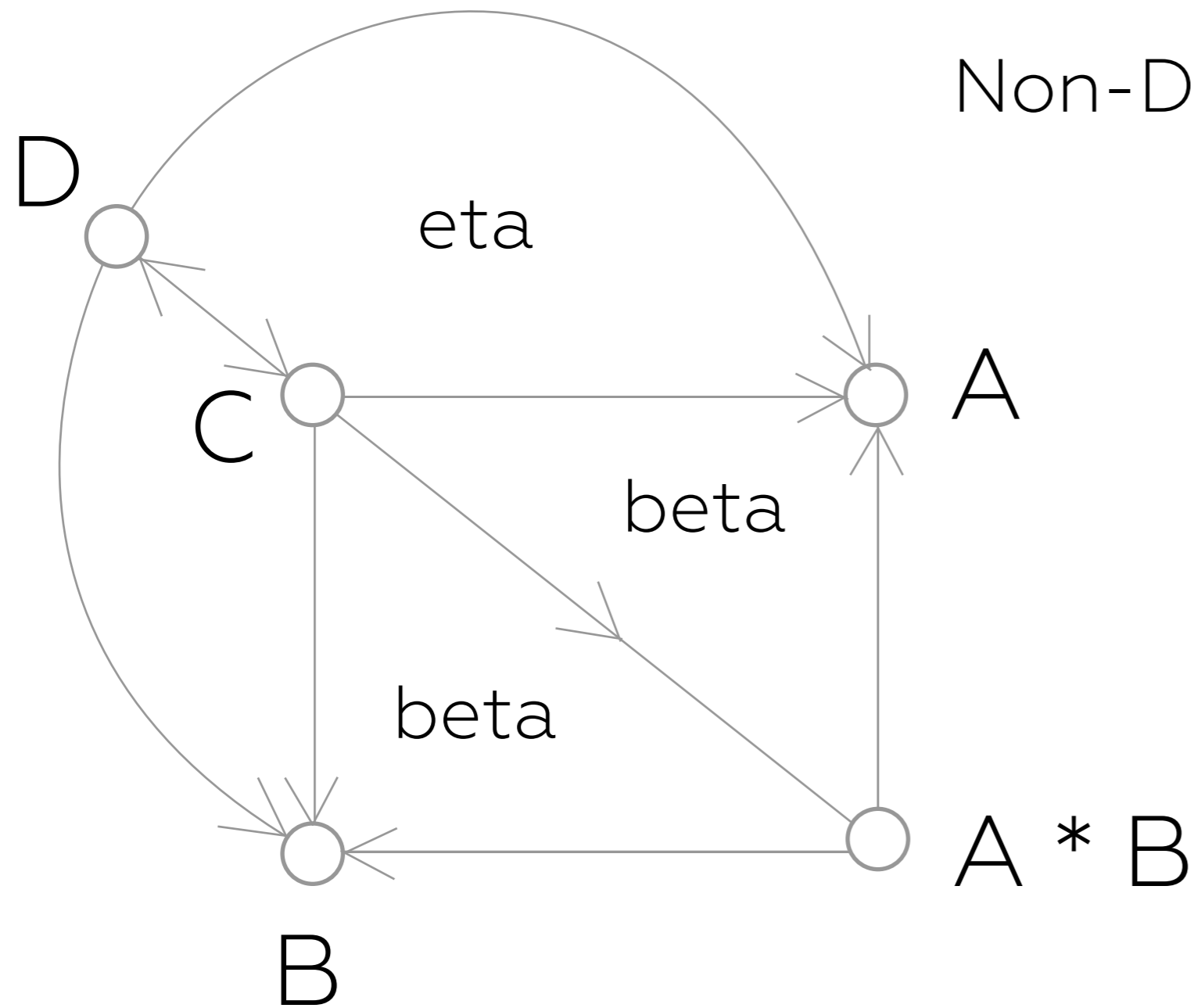
(a,b)

$.1, .2$

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

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

$S (n: \text{nat}) = U_n$

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

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

1. Formation	<code>data nat</code>	<code>data list</code>	<code>Path A x y</code>	<code>data W</code>
2. Introduction	<code>zero, succ</code>	<code>nil, cons</code>	<code>refl A x</code>	<code>sup</code>
3. Elimination	<code>natInd</code>	<code>listInd</code>	<code>J</code>	<code>wInd</code>
4. Computational	<code>Beta, Eta</code>	<code>Beta, Eta</code>	<code>Beta, [Eta]</code>	<code>Beta, Eta</code>

Intuitionistic Propositional Logic

According to Brouwer–Heyting–Kolmogorov interpretation

\forall	\exists	Path	0	1	+
$x:A \rightarrow B(x)$	$x:A * B(x)$	$x:A = y:A$	data empty	data unit	data either
$\lambda (x: A) \rightarrow B(x)$	$(x, B(x))$	refl A x		tt	inl, inr
$f a = B(a)$	pr1, pr2	J	elim0	elim1	eitherInd
Beta, Eta	Beta, Eta	Beta	Beta, Eta	Beta, Eta	Beta, Eta

Proto (Prelude)

For run-time and I/O applications

maybe

$U \rightarrow U$

nothing, just

maybeInd

Beta, Eta

either

$U \rightarrow U \rightarrow U$

inl, inr

eitherInd

Beta, Eta

stream

$U \rightarrow U$

cons

streamInd

Beta, Eta

bool

U

true, false

boolInd

Beta, Eta

vector

$\text{nat} \rightarrow U$

vz, vs

vecInd

Beta, Eta

fin

$\text{nat} \rightarrow U$

fz, fs

finInd

Beta, Eta

Homework: Add I/O interface for finite and infinite loop.

Induction Principle

Natural Numbers Example

`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)

Homework: Add HITs elements as Path Equalities to data declarations.

Hint: use your imagination.

Pi Type : Definition

Family of Types, Fibrations, Fiber Space $A \rightarrow U$, Fiber $B(x)$,
Section $b(x)$, Space of Sections $\Pi(A, B)$

Syntax

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

$T ::= \#identifier$

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

$O_1 ::= U \mid T \mid (O) \mid O O \mid O \rightarrow O$
 $\mid \lambda (l: O) \rightarrow O \mid (l: O) \rightarrow O$

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

Universal Quantifier

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

$\text{lamba } (A : U) (B : A \rightarrow U) (a : A) (b : B a) : A \rightarrow B a = \lambda (x : A) \rightarrow b$

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

$\text{Beta } (A:U) (B:A \rightarrow U) (a:A) (f : A \rightarrow B a) : \text{Path } (B a) (\text{app } A B a (\text{lam } A B a (f a))) (f a)$

$\text{Eta } (A : U) (B : A \rightarrow U) (a : A) (f : A \rightarrow B a) : \text{Path } (A \rightarrow B a) f (\lambda (x:A) \rightarrow f x)$

One beta rule and one eta rule for Pi types.

Sigma Type : Definition

Total Space $\text{Sigma}(A,B)$, Point in Base with Section (a,b)

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

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

$\text{Sigma } (A : U) (B : A \rightarrow U) : U = (x : A) * B x$

$\text{pair } (A : U) (B : A \rightarrow U) (a : A) (b : B a) : \text{Sigma } A B = ?$

$\text{pr1 } (A : U) (B : A \rightarrow U) (x : \text{Sigma } A B) : A = ?$

$\text{pr2 } (A : U) (B : A \rightarrow U) (x : \text{Sigma } A B) : B (\text{pr1 } A B x) = ?$

$\text{Beta1 } (B : A \rightarrow U) (a : A) (b : B a) \rightarrow \text{Path } A a (\text{pr1 } A B (\text{pair } A B a b))$

$\text{Beta2 } (B : A \rightarrow U) (a : A) (b : B a) \rightarrow \text{Path } (B a) b (\text{pr2 } A B (a, b))$

$\text{Eta } (B : A \rightarrow U) (p : \text{Sigma } A B) \rightarrow \text{Path } (\text{Sigma } A B) p (\text{pr1 } A B p, \text{pr2 } A B p)$

$\text{sigRec } (A : U) (B : A \rightarrow U) (C : U) (g : (x : A) \rightarrow B(x) \rightarrow C) (p : \text{Sigma } A B) : C = g p.1 p.2$

$\text{sigInd } (A : U) (B : A \rightarrow U) (C : \text{Sigma } A B \rightarrow U)$

$(p : \text{Sigma } A B) (g : (a : A) (b : B(a)) \rightarrow C(a, b)) : C p = g p.1 p.2$

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 $(A: U): U$

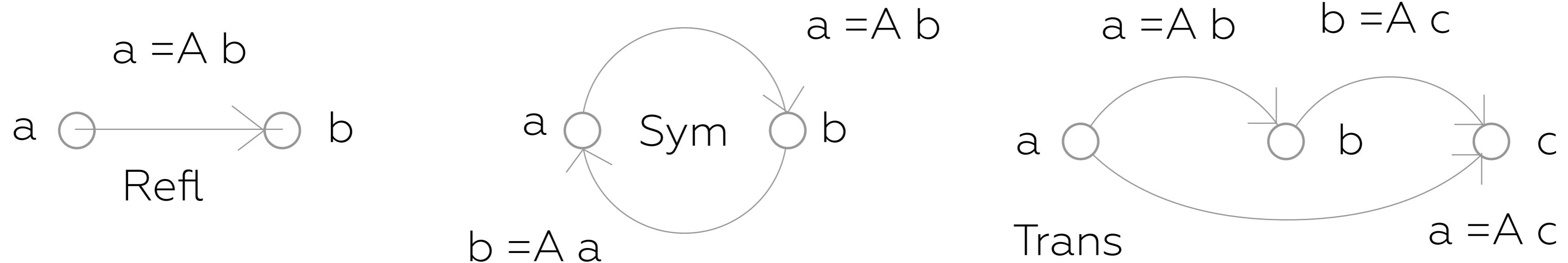
= (Carrier: A)

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

* (Refl: $(x: A) \rightarrow$ Equ x x)

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

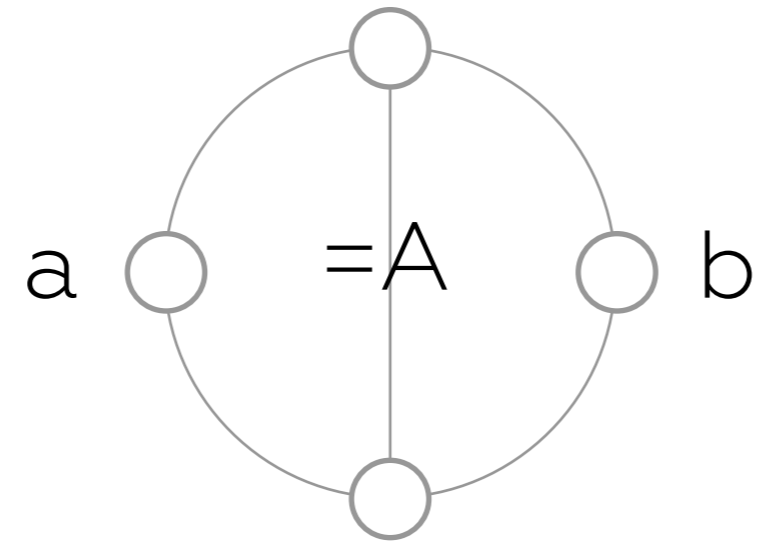
* (Sym: $(x_1, x_2: A) \rightarrow$ Equ $x_1\ x_2 \rightarrow$ Equ $x_2\ x_1$)



Globular Theory

Multidimensional Equality

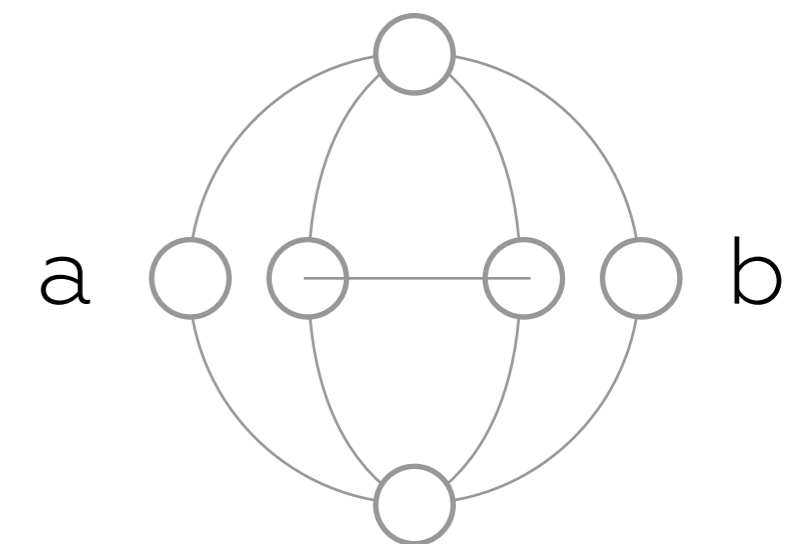
$a =_A b$



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

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

$a =_A b$



$a =_A b$

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

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

$a : A \quad \circ \longrightarrow \triangleright \circ \quad b : A$

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

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)`

HIT: Homotopy

Intro and Elim

```
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)
```

n-Types

Infinity Groupoids

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

Set Theory Theorems

$$\begin{aligned} \text{ac} \quad & (A \ B: U) (R: A \rightarrow B \rightarrow U): (p: (x:A) \rightarrow (y:B) * (R \ x \ y)) \rightarrow (f:A \rightarrow B) * ((x:A) \rightarrow R(x)(f \ x)) \\ & = \backslash(g: (x:A) \rightarrow (y:B) * (R \ x \ y)) \rightarrow (\backslash(i:A) \rightarrow (g \ i).1, \backslash(j:A) \rightarrow (g \ j).2) \end{aligned}$$

$$\begin{aligned} \text{total} \quad & (A:U) (B \ C : A \rightarrow U) (f : (x:A) \rightarrow B \ x \rightarrow C \ x) (w:\text{Sigma } A \ B) : \text{Sigma } A \ C \\ & = (w.1, f (w.1) (w.2)) \end{aligned}$$

Prop Logic

Set Theory Theorems

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)

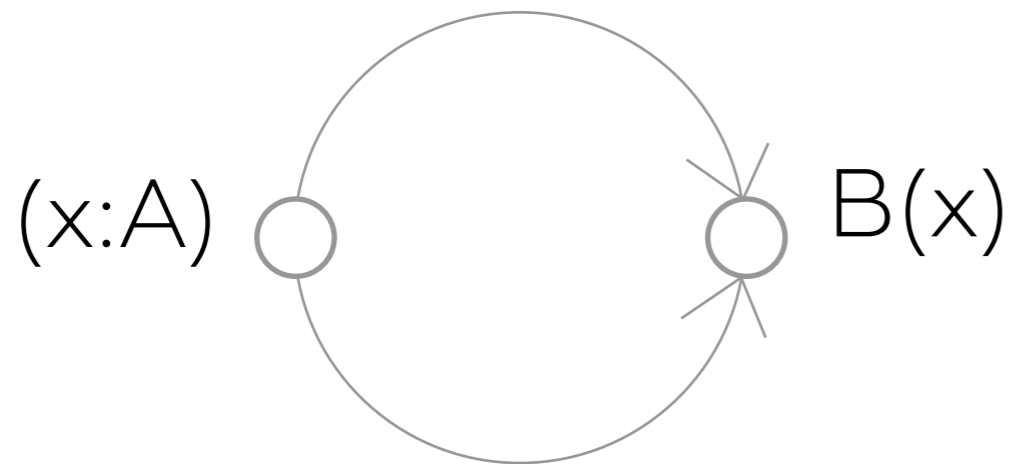
propDec (A : U) (a: isProp A) : isProp (dec A)
propAnd (A B : U) (a: isProp A) (b: isProp B) : isProp (`prod` A B)
propOr (A B : U) (a: isProp A) (b: isProp B) (x: A -> neg B) : isProp (`either` A B)
propNeg (A : U) : isProp (neg A)
propN0 : isProp `empty`

FunExt

Syntax and Model

```
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))
```

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



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

$f = (A \rightarrow B) g$



$\langle i \rangle \backslash (a:A) \rightarrow p a @ i$

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: $F \rightarrow E \rightarrow B$

Moebius $E = S^1$ 'twisted *' $[0,1]$

Trivial: $E = B * F$

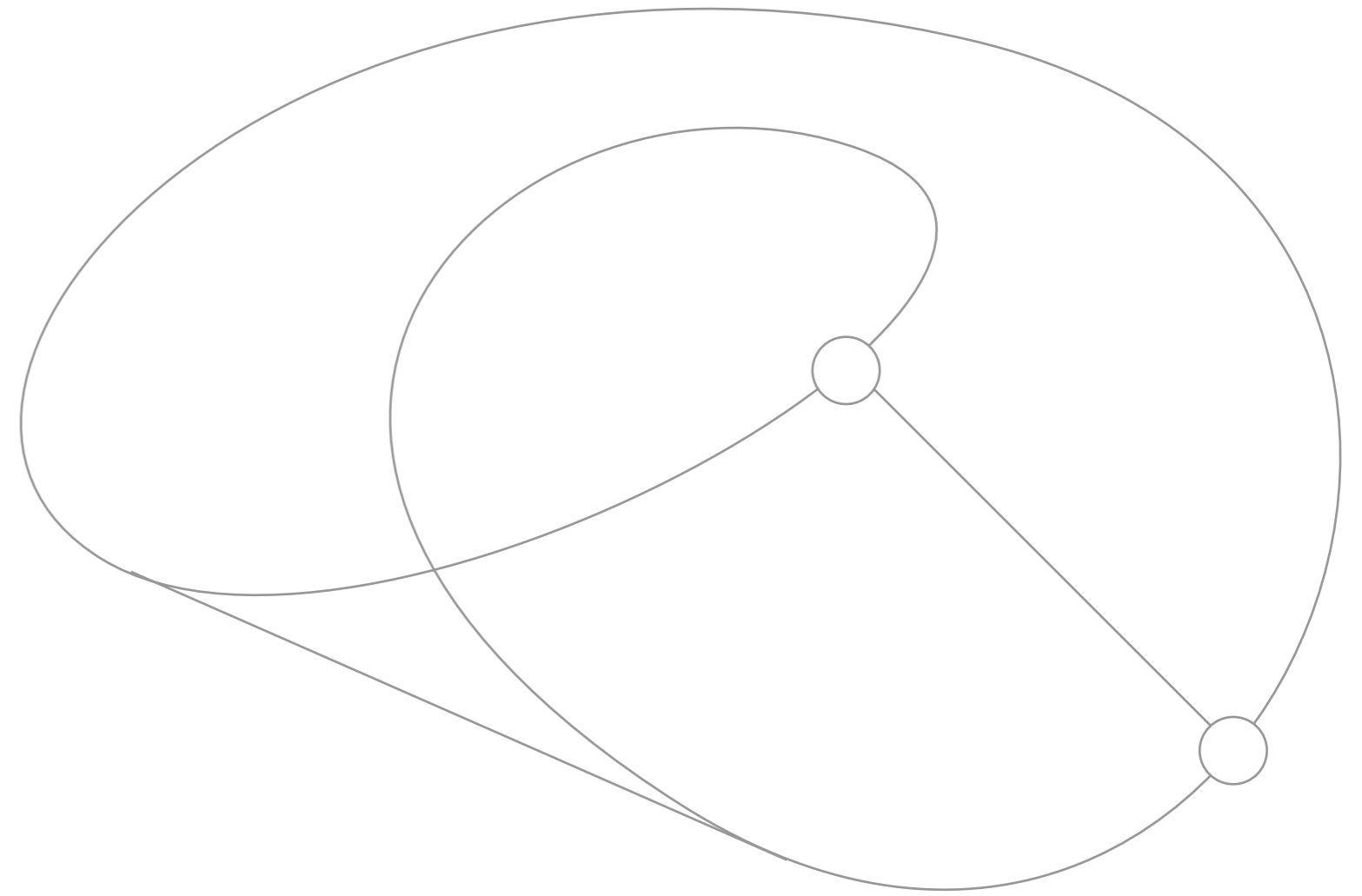
$p : \text{total} \rightarrow B$

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

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

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)
* (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

$ua (A B : U) : U = equiv A B \rightarrow Path U A B$

$uaIntro (A B : U) : ua A B$

$uaElim (A B : U) (p : Path U A B) : equiv A B$

$uaComp (A B : U) (p : Path U A B)$

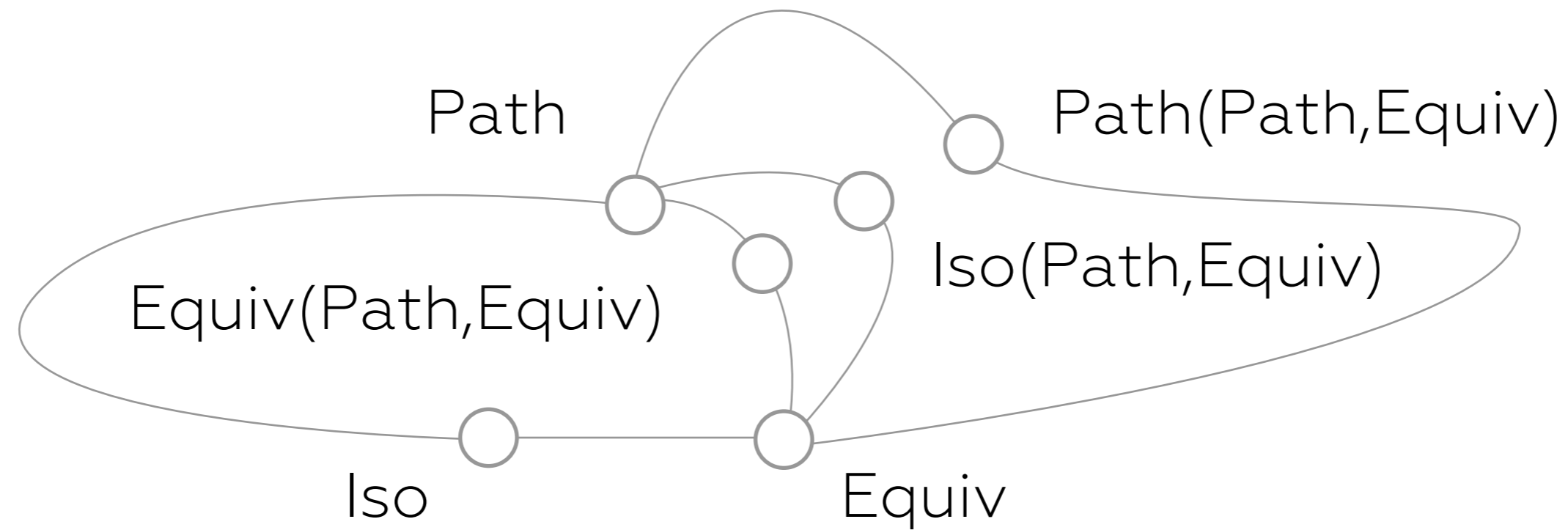
$: Path (Path U A B) (uaIntro A B (uaElim A B p)) p$

$uaUniqueness (A B : U) (w : equiv A B)$

$: Path (A \rightarrow B) w.1 (uaElim A B (uaIntro A B w)).1$

Univalence Axiom

All Equalities Should Be Equal



```

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]

```

Trivial Fiber = Pi

```

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

Category Theory

Categories

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

Category Theory

Functors

`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

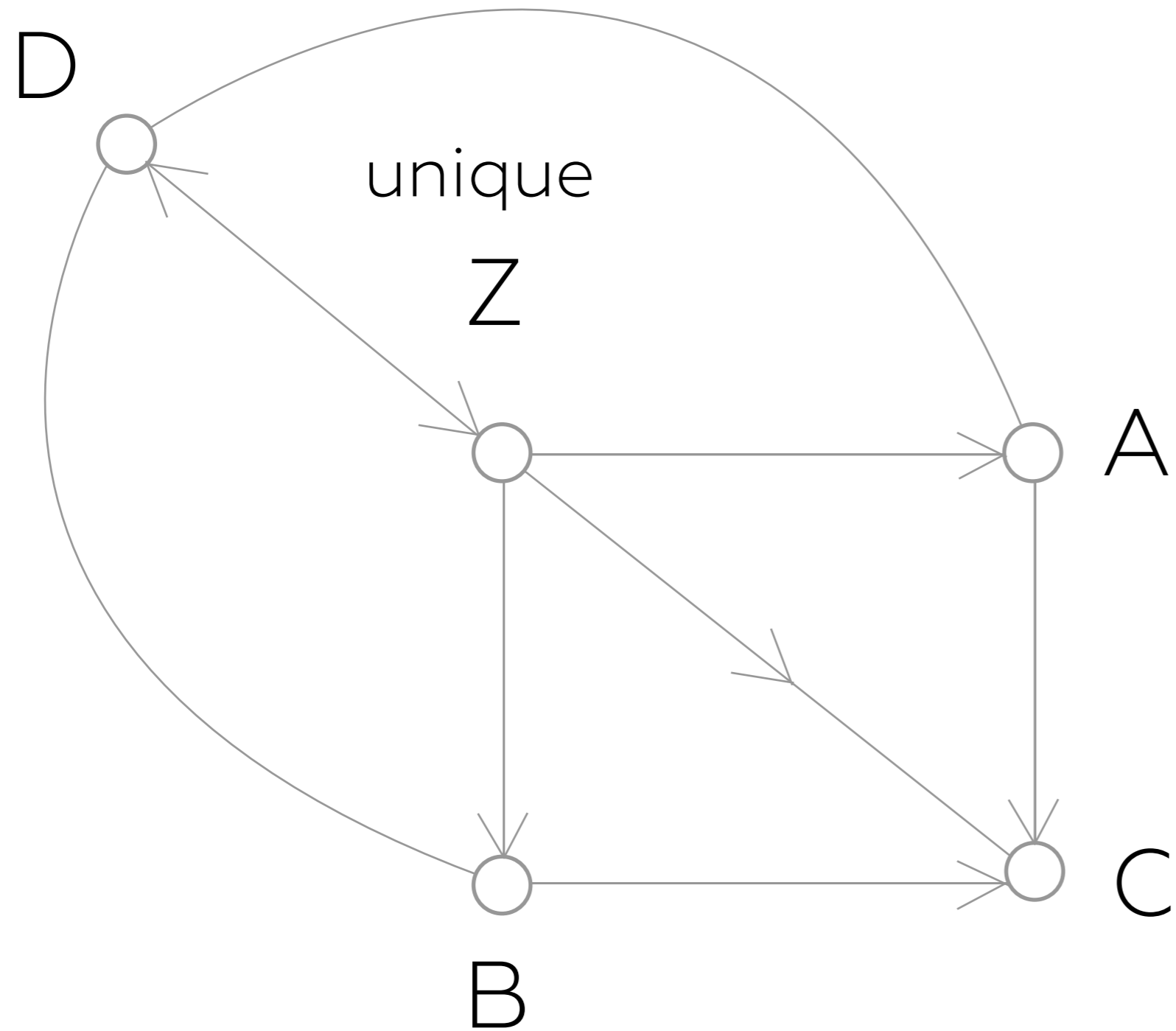
Q (A B C D: Ob) (f: Hom A B) (g: Hom B C) (h: Hom C D)

: `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



Examples:
Products, Fibers

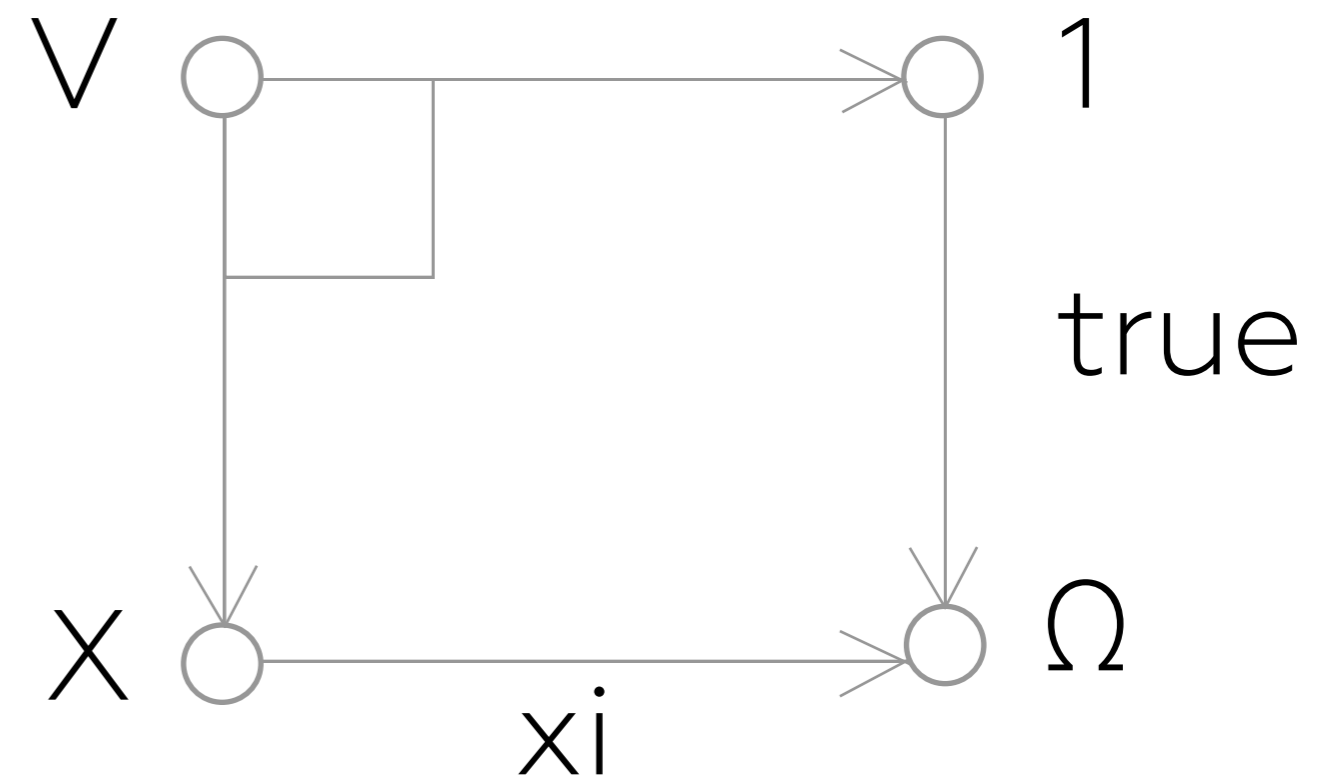
Dual Examples (Pushout):
Coproducts, Cofibers

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

Structures

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: $U = (X: \text{SET}) * \text{isMonoid } X$

cmonoid: $U = (X: \text{SET}) * \text{isCMonoid } X$

group: $U = (X: \text{SET}) * \text{isGroup } X$

abgroup: $U = (X: \text{SET}) * \text{isAbGroup } X$

ring: $U = (X: \text{SET}) * \text{isRing } X$

abring: $U = (X: \text{SET}) * \text{isAbRing } X$

monoidhom (a b: monoid): U
= (f: a.1.1 \rightarrow b.1.1)
* (ismonoidhom a b f)

cmonoidhom (a b: cmonoid): $U = \text{monoidhom } (a.1, a.2.1) (b.1, b.2.1)$

grouphom (a b: group): $U = \text{monoidhom } (a.1, a.2.1) (b.1, b.2.1)$

abgrouphom (a b: abgroup): $U = \text{monoidhom } (a.1, a.2.1.1) (b.1, b.2.1.1)$

cmonabgrouphom (a: cmonoid) (b: abgroup): $U = \text{monoidhom } (a.1, a.2.1) (b.1, b.2.1.1)$

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)
```

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

cubical: Resolver.hs:(293,26)-

(316,29): Non-exhaustive patterns

in case

Mahlo Universe

Structures

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) ->

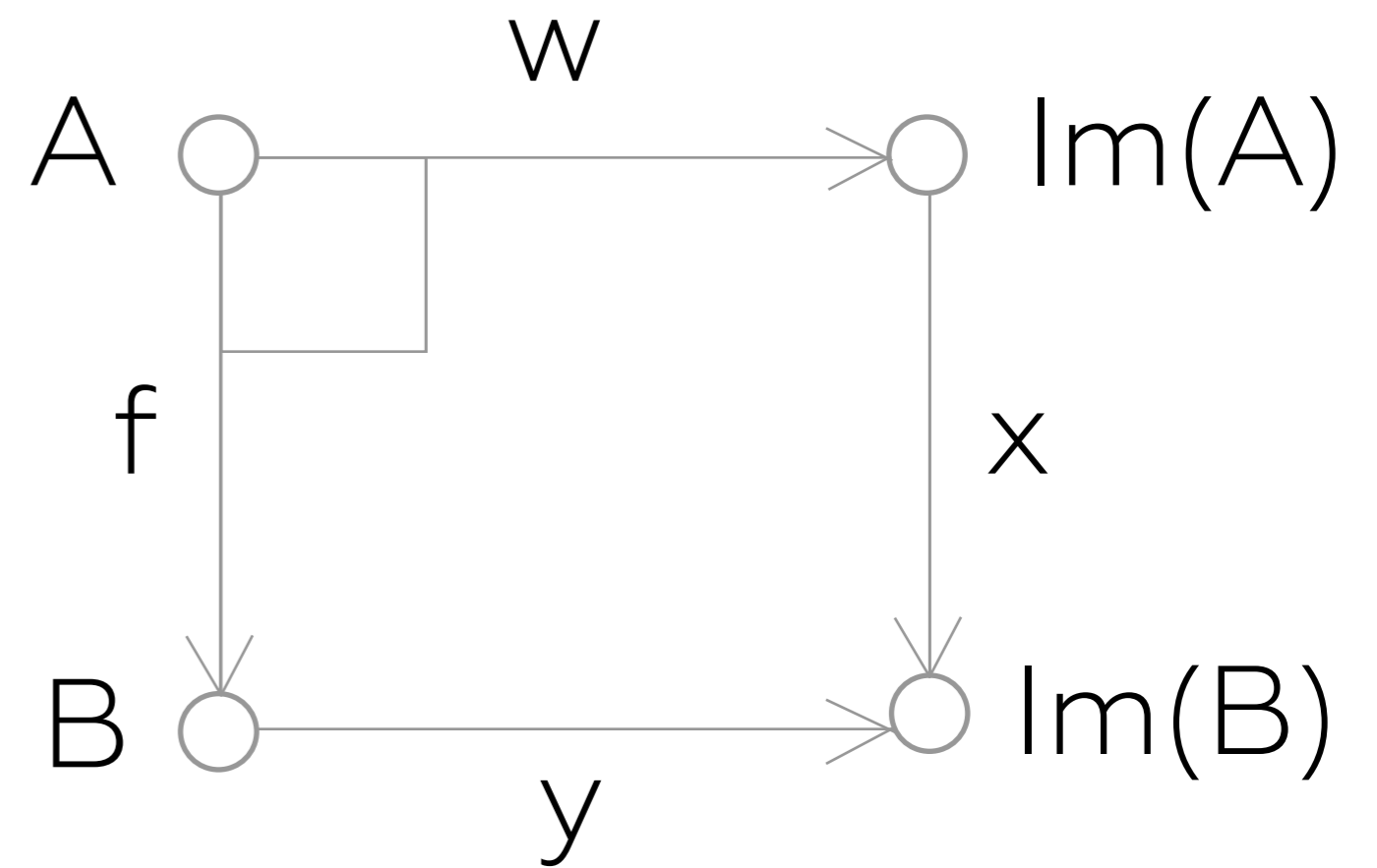
Universe f g -> V = undefined

Differential Topology

Étale Maps

$\text{ÉtaleMap } (A B: U): U$
 $= (f: A \rightarrow B)$
 $* \text{isÉtaleMap } A B f$

$\text{isÉtaleMap } (A B: U) (f: A \rightarrow B): U$
 $= \text{isPullbackSq } A iA B (\text{Im } B) x y w f h$ *where*
 $iA : U = \text{Im } A$
 $iB : U = \text{Im } B$
 $x: iA \rightarrow iB = \text{ImApp } A B f$
 $y: B \rightarrow iB = \text{ImUnit } B$
 $w: A \rightarrow iA = \text{ImUnit } A$
 $c1: A \rightarrow iB = \circ A iA iB x w$
 $c2: A \rightarrow iB = \circ A B iB y f$
 $T2: U = (a:A) \rightarrow \text{Path } iB (c1 a) (c2 a)$
 $h: T2 = \backslash(a : A) \rightarrow \langle i \rangle \text{ImNaturality } A B f a @ -i$



Differential Topology

Manifolds

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

$\text{Im} : U \rightarrow U = \text{undefined}$

$\text{ImUnit} (A: U) : A \rightarrow \text{Im} A = \text{undefined}$

$\text{isCoreduced} (A:U): U = \text{isEquiv} A (\text{Im} A) (\text{ImUnit} A)$

$\text{ImCoreduced} (A:U): \text{isCoreduced} (\text{Im} A)$

$\text{ImApp} (A B: U) (f: A \rightarrow B): \text{Im} A \rightarrow \text{Im} B$

$= \text{ImRecursion} A (\text{Im} B) (\text{ImCoreduced} B) (\circ A B (\text{Im} B) (\text{ImUnit} B) f)$

$\text{ImNaturality} (A B:U) (f:A \rightarrow B): (a:A) \rightarrow \text{Path} (\text{Im} B)((\text{ImUnit} B)(f a))((\text{ImApp} A B f)(\text{ImUnit} A a))$

$\text{ImInduction} (A:U)(B:\text{Im} A \rightarrow U)(x: (a: \text{Im} A) \rightarrow \text{isCoreduced}(B a))$

$(y:(a: A) \rightarrow B(\text{ImUnit} A a)):(a:\text{Im} A) \rightarrow B a$

$\text{ImComputeInduction} (A:U)(B:\text{Im} A \rightarrow U) (c:(a:\text{Im} A) \rightarrow \text{isCoreduced}(B a))$

$(f:(a:A) \rightarrow B(\text{ImUnit} A a))(a:A)$

$: \text{Path} (B (\text{ImUnit} A a)) (f a) ((\text{ImInduction} A B c f) (\text{ImUnit} A a))$

HIT: 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 -> bool
```

```
  succ x -> susp (S x)
```

<https://groupoid.space/mltt/hopf/>

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: abgroup_hom (chain zero) head)

* ((n: nat) -> abgroup_hom (chain (succ n)) (chain n))

CochainComplex: U

= (head: abgroup)

* (cochain: nat -> abgroup)

* (augment: abgroup_hom head (cochain zero))

* ((n: nat) -> abgroup_hom (cochain n) (cochain (succ n)))

Impredicative Encoding

As in version of Steve Awodey, HITs encoding

$\text{Nat_Church} = (X: U) \rightarrow (X \rightarrow X) \rightarrow X \rightarrow X$

$\text{Nat} = (X: U) \rightarrow \text{isSet } X \rightarrow (X \rightarrow X) \rightarrow X \rightarrow X$

$\text{Unit} = (X: U) \rightarrow \text{isSet } X \rightarrow X \rightarrow X$

$1 = (\text{one}: \text{Unit}) * ((X \ Y: U) (x: \text{isSet } X) (y: \text{isSet } Y) (f: X \rightarrow Y) \rightarrow \text{naturality } X \ Y \ f \ (\text{one } X \ x) \ (\text{one } Y \ y))$

$N = (\text{one}: \text{Nat}) * ((X \ Y: U) (x: \text{isSet } X) (y: \text{isSet } Y) (f: X \rightarrow Y) \rightarrow \text{naturality } X \ Y \ f \ (\text{one } X \ x) \ (\text{one } Y \ y))$

Truncation $\|A\|$ parametrized by $(A:U)$ type $= (X: U) \rightarrow \text{isProp } X \rightarrow (A \rightarrow X) \rightarrow X$

$S^1 = (X:U) \rightarrow \text{isGroupoid } X \rightarrow (x:X) \rightarrow \text{Path } X \ x \ x \rightarrow X$

Arbitrary $(A:U)$ type $= (X: U) \rightarrow \text{isSet } X \rightarrow (A \rightarrow X) \rightarrow X$

Impredicative Encoding

Encode Unit. Homework: Bool, Circle, Sphere

$\text{unitEnc}' : U = (X : U) \rightarrow \text{isSet } X \rightarrow X \rightarrow X$

$\text{naturality } (X Y : U)(f : X \rightarrow Y)(a : X \rightarrow X)(b : Y \rightarrow Y) : U = \text{Path } (X \rightarrow Y)(o \ X \ X \ Y \ f \ a)(o \ X \ Y \ Y \ b \ f)$

$\text{isUnitEnc } (\text{one} : \text{unitEnc}') : U$

$= (X Y : U)(x : \text{isSet } X)(y : \text{isSet } Y)(f : X \rightarrow Y) \rightarrow \text{naturality } X \ Y \ f \ (\text{one } X \ x)(\text{one } Y \ y)$

$\text{unitEnc} : U = (x : \text{unitEnc}') * \text{isUnitEnc } x$

$\text{unitEncStar} : \text{unitEnc} = (\backslash(X : U)(_ : \text{isSet } X) \rightarrow \text{idfun } X, \backslash(X Y : U)(_ : \text{isSet } X)(_ : \text{isSet } Y) \rightarrow \text{refl}(X \rightarrow Y))$

$\text{unitEncRec } (C : U) (s : \text{isSet } C) (c : C) : \text{unitEnc} \rightarrow C = \backslash(z : \text{unitEnc}) \rightarrow z.1 \ C \ s \ c$

$\text{unitEncBeta } (C : U) (s : \text{isSet } C) (c : C) : \text{Path } C (\text{unitEncRec } C \ s \ c \ \text{unitEncStar}) \ c$

$\text{unitEncEta } (z : \text{unitEnc}) : \text{Path } \text{unitEnc} \ \text{unitEncStar } z$

$\text{unitEncInd } (P : \text{unitEnc} \rightarrow U) (a : \text{unitEnc}) : P \ \text{unitEncStar} \rightarrow P \ a$

$\text{unitEncCondition } (n : \text{unitEnc}') : \text{isProp } (\text{isUnitEnc } n)$

Literature Overview

groupoid.space/mltt/infinity

Lof72 Voevodsky15 Dybjer08 Hofmann94 Lof84 Sozeau Clairambault05
Jacobs99 Coq88 Selsam16 Abel08 Joyal14 Hofmann96 Bohm85 Seely84
Henk93 Pfenning89 Curien14 [HoTT13](#) Mortberg17 Erik97 Wadler90
Castellan14 Shulman15 Hedberg98 Gambino03 Voevodsky14 Orton17
Hermida95 Dybjer94 Dybjer95 Huber16 Curien08 Jacobs97 Bishop67
Huber17 MacLane71 Vene00 Nordstrom90 Angiuli16 Lawvere09 Basold16
Hermida98 Barthe00

<https://github.com/groupoid/cafe>

Thank You!

Maxim Sokhatsky

<https://groupoid.space>