ΠΣ

1

234567

$\mathsf{FF}_\omega{}^8$

https://5ht.github.io/bertrand/

https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf

http://www-users.math.umn.edu/~arnold/disasters/ariane5rep.html

http://mdbailey.ece.illinois.edu/publications/imc14-heartbleed.pdf

https://arxiv.org/pdf/1809.03981.pdf

https://www.cs.umd.edu/~aseem/solidetherplas.pdf
https://www.dcs.ed.ac.uk/home/mlj/

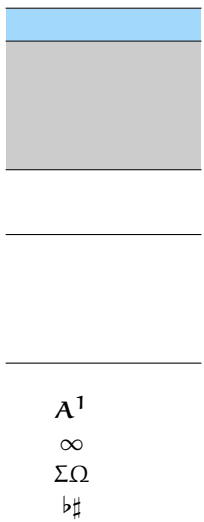
ω

ΠΣ

$\star : \star\star : \Box\Box : \Box\Box : \star\mathrm{P}_\omega$

$$\begin{array}{ccc} \lambda_\omega & \longrightarrow & \lambda\mathrm{P}_\omega \\ \lambda_2 & \longrightarrow & \lambda\mathrm{P}_2 \\ \downarrow & \ddots & \downarrow \\ \lambda_\to & \longrightarrow & \lambda\mathrm{P}_\underline{\omega} \end{array}$$

$[0, 1]$



$\mathbf{A}^1$

∞

$\Sigma\Omega$

$\flat\sharp$

910

1112

1314

https://groupoid.github.io/languages
https://arxiv.org/pdf/1804.07608.pdf
https://n2o.dev/ua
https://anders.groupoid.space/lib

15 16

https://n2o.dev
https://groupoid.space

12

34

567

http://mrg.doc.ic.ac.uk/kohei/

89

Henk
1011

$\pi\pi$

$O_{CPS}\pi$Bob
$O_{CPS}{}^{1213}$Joe

https://5ht.co/ctt.pdf
https://5ht.co/cctt.pdf
https://web.sfc.keio.ac.jp/~hagino/thesis.pdf
https://github.com/devaspot/charity
http://nickbenton.name/coqasm.pdf
https://www.cl.cam.ac.uk/~mom22/tpholls09-lisp.pdf

$$O_{\lambda}\lambda O_{\pi}\pi O_{\mu}O_{\prod}O_{\Sigma}O_{=}O_{W}O_{I}$$

1

ΠΣ=+⊥⊤NUᵢEl

https://github.com/guillaumebrunerie/initiality

$$\Pi x : op(o) : \amalg p : concept(o)$$

$$Ob : \amalg Hom : Ob \to Ob \to \amalg$$

$$id\circ$$

$$A : Ob(c) isContr(\Sigma(B : Ob(C)), A = B)$$

$$p : E \to Bf : Y \to Xp(f)Bp(f) = id_B$$

$$p : E \to Bf : Y \to XEt = p(f)u : Z \to Yp(f) = p(u)a : Z \to Yf \circ a = u$$

$$p : E \to Bf : Y \to XEt = p(f)u : K \to JBv : Z \to XEp(v) = t \circ uw : Z \to Y Ev = f \circ wp(w) = u$$

$$B^{\to}Ba : Ob_B^{\to} = Hom_B(x, y)Hom_B^{\to} = [f : Hom_B, g : Hom_B]B$$

$$\begin{array}{ccc} x & \xrightarrow{f} & f(x) \\ a \downarrow & & \downarrow f(a) \\ y & \xrightarrow{g} & g(y) \end{array}$$

$$p : E \to BBu : J \to IBX \in p(I)Bf : Y \to XuXu$$

$$p^{-1}B^{op} \to CatPsh(B) = [B^{op}, Cat]$$

$$p : X \to Bq : Y \to BBF : X \to Yq \circ F = PxXpF(x)q$$

$$p : E \to Bq : D \to ABFib_B(p, q)Cat/Bp \to q(H : E \to D, K : B \to A)fp H(f)q$$

$$p : E \to B^{\to}cod \circ p : E \to Bf \in EpfB$$

$$Psh(B)Fib(B)$$

$$\int : Psh(B) \xrightarrow{\cong} Fib(B)$$

$\Pi\Sigma$

$$\mathrm{Ob}_{\mathrm{C}}\Pi(A,B)\mathrm{Hom}_{\mathrm{C}}(\Pi(A,B),\Pi(A',B')[f : A \to A', g(x : A) : B(x) \to B'(f(x))$$

```
def CwF : U := Σ (C: precategory) (T: catfunctor C Fam)
    (context: isContext C) (terminal: isTerminal C), isComprehension C T
```

$$CF : C^{op} \to \mathbf{Set} \mathbf{C} \mathbf{Set}$$

$$Ct \in CTy, Tm : C^{op} \to \mathbf{Set} p : Tm \to Ty$$

```
def naturalModel : U := Σ (C  : precategory) (_  : isCategory C)
    (t : terminal C) (Tm : carrier C) (Ty : carrier C)
    (p : hom C VT V), Π (f : homTo C V), hasPullback C (Tm, f, Ty, p)
```

$$C - C -$$

$$P, Q : C^{op} \to \alpha : Q \to P \alpha Ob(C) x : Ob(C) p_x : D \to Cy : Q(D)$$

$$y \overset{y(D) \overset{\psi}{\to} Q}{\underset{y(C) \overset{p_x}{\to} P}{\overset{\nearrow}{\underset{\searrow}{}}}} \alpha$$

$$F : C \to D \phi_{Ty} : F_! Ty_C \to Ty_D \phi_{Tm} : F_! Tm_C \to Tm_D$$

$$F_! Tm(C) \overset{\phi_{Tm}}{\to} Tm(D)$$
$$F_! p_C \downarrow \qquad \downarrow p_D$$
$$F_! Ty(C) \underset{\phi_{Ty}}{\to} Ty(D)$$

$$F_! : C^{op} \to D^{op}$$

O_λ	λ
O_π	π
O_μ	
O_Π	
O_Σ	
$O_=$	
O_W	
O_I	
$O_\triangleright$	π
$O_/$	
O_H	
$O_\dashv$	

TmTy

$\beta\eta$

$O_\Pi O_\Sigma O_= O_{PTS} O_{MLTT-80} O_{HTS}$

$$O_\Pi \to O_\Sigma \to O_= \to O_W \to O_I.$$

$$O_{PTS}(O_\Pi) \to O_{MLTT-72}(O_\Pi, O_\Sigma) \to O_{MLTT-75}(.., O_\Sigma, O_=) \to$$
$$\to O_{MLTT-80}(..., O_=, O_W) \to O_{HTS}(..., O_W, O_I).$$

$O_{PTS} : O_\Pi \to U$
$O_{MLTT-72} : O_\Pi \to O_\Sigma \to U$
$O_{MLTT-75} : O_\Pi \to O_\Sigma \to O_= \to U$
$O_{MLTT-80} : O_\Pi \to O_\Sigma \to O_= \to O_W \to U$
$O_{HTS} : O_\Pi \to O_\Sigma \to O_= \to O_W \to O_I \to U$

$O_{HTS} = O_{\Pi\Sigma=WI}$

$$O_{HTS} = O_{\Pi\Sigma=WI} : O_\Pi \to O_\Sigma \to O_= \to O_W \to O_I \to U.$$

$O_{MLTT-80}$

$\beta\eta\beta\eta$

$$O_\infty : O_{CPS} \to O_{PTS} \to O_{MLTT-80} \to O_{HTS} \to \dots$$

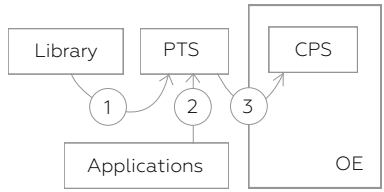
$f : O_x \to O_y O_x O_y F_O$

$\mathcal{F}_{\mathcal{L}} \mathcal{F} f : L \to L', L \supseteq L \mathcal{F} f : L \to L', L \subseteq L \mathcal{F} f : L \to L', L' \cong L \mathcal{F} f : L \to O \mathcal{F} f : L \to L' L' \mathcal{F} f : L \to L' L'$

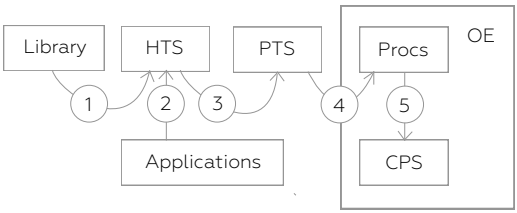
$O_{I*} \to O_{\Pi=} O_\Pi \to O_{\Pi\Sigma} O_\Pi \to O_{\Pi\Sigma} O_{\Pi*} \to O_\Pi$

```
Henk = U^n, Π.
Frank = U^n, Π, Ind.
Errett = U^n, Π, Σ, Prop.
Per = U^n, Π, Σ, 0, 1, 2, W, Prop.
Christine = U^n, Π, Σ, Id, Ind, Prop
Anders = U^n, V^n, Π, Σ, 0, 1, 2, W, Path, Prop.

Urs = Anders, U_i^|o|, A × B, G → A, ♮, ♭, ♯, ℑ, ◯.
Dan = Anders, Chain, Cochain, Category, Monoid, Group, Ring, Δ.
Fabien = Anders, k, A^1, S^1,1, L_A^1, Susp, Trunc^n, Nisn, K^1(Z, n), BGL, MGL.
Jack = Anders, Fib^n, Susp, Trunc^n, N, N_∞, Spec, π_n^S(A), S^o[p], Group,
       A ∪ B, [A, B], H^n(X; G), G ⊗ H, SS(E, r).
```

$$\mathrm{PTS}_{\mathrm{CPS}} = \begin{cases} \mathrm{Ob} : \{\mathrm{O}_{\mathrm{CPS}}, \mathrm{O}_{\mathrm{PTS}}\} \\ \mathrm{Hom} : \{1, 2 : \mathbb{1} \to \mathrm{O}_{\mathrm{PTS}}, 3 : \mathrm{O}_{\mathrm{PTS}} \to \mathrm{O}_{\mathrm{CPS}}\} \end{cases}$$



$$\mathrm{Total} = \begin{cases} \mathrm{Ob} : \{\mathrm{O_{CPS}}, \mathrm{O_{PTS}}, \mathrm{O_{MLTT-75}}, \mathrm{O_{MLTT-80}}, \mathrm{O_{HTS}}\} \\ \mathrm{Hom} : \begin{cases} 1, 2 : \mathbb{1} \to \mathrm{O_{HTS}}, 3 : \mathrm{O_{MLTT-75}} \to \mathrm{O_{MLTT-80}} \\ 4 : \mathrm{O_{HTS}} \to \mathrm{O_{MLTT-80}}, 5 : \mathrm{O_{MLTT-80}} \to \mathrm{O_{PTS}}, 6 : \mathrm{O_{PTS}} \to \mathrm{C} \end{cases} \end{cases}$$



O_{CPS}

$$O_{CPS} = \begin{cases} Ob : \{maybeCPS\} \\ Hom : \{eval : Ob \to Ob\} \end{cases}$$

$O_\lambda O_\pi O_\mu$

O_{CPS}

```
def CPS : U
 := inductive { lambda (c: Joe CPS)
              | process (m: Bob CPS)
              | tensor (f: Alice CPS)
              }
```

$\lambda O_{CPS} maybe$

$$O_{CPS} : O_\lambda \to O_\pi \to O_\mu \to U$$

O_λ

```
def Joe (cps: U): U
 := inductive { var (x: nat)
              | lam (l: nat) (d: cps)
              | app (f a: cps)
              }
```

O_π

```
def Bob (lang: U) : U
 := inductive { process (protocol: lang)
              | spawn (cursors: lang) (core: nat) (program: lang)
              | snd (cursor: lang) (data: lang)
              | rcv (cursor: lang)
              | pub (size: nat)
              | sub (cursor: lang)
              }
```

O_μ

```
def Alice (lang: U) : U
 := inductive { Variable (_: Var)
              | Prim (_: Builtin)
              | Star | True | False
              | Int (_: nat) | Float (_: float)
              | Lambda (a: Var) (b: Linear) (c: Exp)
              | App (a b: Exp)
              | Pair (a b: Var) (c d: Exp)
              | Consume (a: Var) (b c: Exp)
              | Gen (a: Var) (b: Exp)
              | Spec (a: Exp) (b: Fraction)
```

```
              | Fix (a b: Var) (c d: Linear) (e: Exp)
              | If (a b c: Exp)
              | Let (a: Var) (b c: Exp)
              }

def Linear : U
 := inductive { Empty | Unit | Bool
              | Int | Float
              | Tensor (a: Fraction) (x: Dimension)
              | Pair (a b: Linear) | Fun (a b: Linear)
              | Consume (a: Linear) | All (a: Var) (b: Linear)
              }

def Builtin : U
 := inductive { Intop (a: Arith) | Floatop (a: Arith)    -- SIMD types
              | Get | Set | Duplicate | Free             -- linearity
              | Transpose | Size                         -- matrices
              | Asum | Axpy | Dotp | Rotm | Scal | Amax  -- BLAS Level 1
              | Symm | Gemm | Syrk | Posv                -- BLAS Level 3
              }

def Fraction  : U := inductive { Z | S (_: Fraction) }
def Dimension : U := inductive { Vector | Matrix | Stream | Table }
def Arith     : U := inductive { Add | Sub | Mul | Div | Eq | Lt | Gt }
```

Π

Σ

$\mathbb{N} \to \mathcal{U}$

O_{PTS}

$$O_{PTS} = \begin{cases} \mathrm{Ob} : \{X : \mathrm{maybePTS}, \mathrm{target} : \mathrm{maybeCPS}\} \\ \mathrm{Hom} : \begin{cases} \mathrm{type}, \mathrm{norm} : X \to X, \mathrm{extract} : X \to \mathrm{target} \\ \mathrm{certify} : X \to \mathrm{target} = \mathrm{type} \circ \mathrm{norm} \circ \mathrm{extract} \end{cases} \end{cases}$$

$O_{PTS} O_{PTS} \Pi$

```
def PTS : U := inductive { forall (_: Pi PTS) }
def Henk := PTS
```

O_{Π}

```
def Pi (lang: U) : U
 := inductive { fibrant (n: nat)
              | variable (x: name) (l: nat)
              | pi (x: name) (l: nat) (f: lang)
              | lambda  (x: name) (l: nat) (f: lang)
              | application (f a: lang)
              }
```

$\Pi\Sigma MLTT-75\Pi\Sigma^2\Pi\forall^3$

$O_{MLTT-75}$

$$O_{MLTT-75} = \begin{cases} Ob : \{maybeMLTT-75\} \\ Hom : \begin{cases} type, norm : Ob \to Ob \\ certify : Ob \to Ob = type \circ norm \end{cases} \end{cases}$$

$O_{MLTT-75}O_{MLTT-75}O_\Pi O_\Sigma O_=$

```
def MLTT : U
 := inductive { forall (_: Pi MLTT)
              | sigma (_: Sigma MLTT)
              | id (_: Id MLTT)
              }
```

$O_\Sigma \Sigma O_{MLTT-72}O_{\Pi\Sigma}$

```
def Sigma (lang: U) : U
 := inductive { sigma (n: name) (a b: lang)
              | pair (a b: lang)
              | fst (p: lang)
              | snd (p: lang)
              }
```

$O_= O_{MLTT-75}O_{\Pi\Sigma=}$

```
def Id (lang: U) : U
 := inductive { identity (t a b: lang)
              | id_intro (a b: lang)
              | id_elim (a b c d e: lang)
              | id_compute (a b c d e: lang)
              }
```

$O_=\eta$

---

https://github.com/zlizta/pisigma-0-2-2
https://github.com/sweirich/pi-forall

$\mathcal{O}_{\mathrm{MLTT-80}}$

$$\mathcal{O}_{\mathrm{MLTT-80}} = \begin{cases} \mathrm{Ob} : \{X : \mathrm{maybePM}, \mathrm{target} : \mathrm{maybeCPS}\} \\ \mathrm{Hom} : \begin{cases} \mathrm{type}, \mathrm{norm}, \mathrm{induction} : X \to X, \mathrm{extract} : X \to \mathrm{target} \\ \mathrm{certify} : X \to \mathrm{target} \\ \mathrm{cerfity} = \mathrm{type} \circ \mathrm{norm} \circ \mathrm{induction} \circ \mathrm{extract} \end{cases} \end{cases}$$

$\mathcal{O}_{=}\mathcal{O}_{\Sigma}\mathcal{O}_{\Pi}\mathcal{O}_{0}\mathcal{O}_{1}\mathcal{O}_{2}\mathcal{O}_{W}$

$\mathcal{O}_{\mathrm{MLTT-80}}$

```
def MLTT-80 := Per
def Per : U
 := inductive { forall (_: Pi Per)
              | sigma (_: Sigma Per)
              | id (_: Id Per)
              | 0 (_: Empty Per)
              | 1 (_: Unit Per)
              | 2 (_: Bool Per)
              | W (_: W Per)
              }

def W (lang: U) : U
 := inductive { W_Form (n: name) (a b: lang)
              | W_Sup (a b: lang)
              | W_Ind (a b c: lang)
              }

def Empty (lang: U) : U := inductive { 0_Ind (a: lang) }
def Unit  (lang: U) : U := inductive { unit | star | 1_Ind (a: lang) }
def Bool  (lang: U) : U := inductive { bool | true | false | 2_Ind (a: lang)
     }
```

O_{CIC}

$$O_{CIC} = \begin{cases} \mathrm{Ob} : \{X : \mathrm{maybePM}, \mathrm{target} : \mathrm{maybeCPS}\} \\ \mathrm{Hom} : \begin{cases} \mathrm{type}, \mathrm{norm}, \mathrm{induction} : X \to X, \mathrm{extract} : X \to \mathrm{target} \\ \mathrm{certify} : X \to \mathrm{target} \\ \mathrm{cerfity} = \mathrm{type} \circ \mathrm{norm} \circ \mathrm{induction} \circ \mathrm{extract} \end{cases} \end{cases}$$

$O_{=}O_{\Sigma}O_{\Pi}$

O_{CIC}

```
def Frank := inductive { forall (_: Pi Frank) | ind (_: Ind Frank) }
def Christine := CIC
def CIC : U
 := inductive { forall (_: Pi CIC)
              | sigma (_: Sigma CIC)
              | id (_: Id CIC)
              | prop (_: Id CIC)
              | ind (_: Ind CIC)
              }
```

$O_{IND}O_{IND}$

```
def Ind (lang: U) : U
 := inductive { formation (_: Inductive lang)
              | constructor (_: list (triple Nat Inductive lang))
              | eliminator (t: Inductive) (a b: lang) (cases: list lang)
              }

def Inductive (lang: U) : U
 := Σ (name : string)
      (params : list (prod name lang))
      (level : Nat)
      (constrs : list (prod (Nat lang))), 1
```

O_{HTS}

$$O_{HTS} = \begin{cases} Ob : \{maybeHTS\} \\ Hom : \begin{cases} type, norm : Ob \to Ob \\ certify : Ob \to Ob = type \circ norm \end{cases} \end{cases}$$

$O_{HTS} O_I O_W O_= O_\Sigma O_\Pi$

```
def HTS : U
 := inductive { forall (_: Pi HTS)
              | sigma (_: Sigma HTS)
              | id (_: Id HTS)
              | prop (_: Id HTS)
              | 0 (_: Empty HTS)
              | 1 (_: Unit HTS)
              | 2 (_: Bool HTS)
              | W (_: W HTS)
              | homotopy (_: Homotopy HTS)
              }
```

$O_{MLTT-80}$

O_I

```
def CCHM (lang: U) : U
 := inductive { pretype (n: nat)
              | PathP (_: lang) | PLam (_: lang) | PApp (f a: lang)
              | I | 0 | 1 | And (a b: lang) | Or (a b: lang) | Neg (_: lang)
              | Transp (a b: lang) | HComp (a b c d: lang)
              | Partial (_: lang) | PartialP (a b: lang) | System (_: lang)
              | Sub (a b c: lang) | Inc (a b: lang) | Ouc (: lang)
              | Glue (: lang) | GlueElem (a b c: lang) | Unglue (_: lang)
              }
```

$O_{HTS} O_= O_I$

```
fun a (0, n) = n + 1
  | a (m, 0) = a (m - 1, 1)
  | a (m, n) = a (m - 1, a (m, n - 1))
```

```
fun proc =
let val p0 = pub(0,8)
    val s1 = sub(0,p0)
    val s2 = sub(0,p0)
 in send(p0,11);
    send(p0,12);
    [ receive(s1);
      receive(s2);
      receive(s1);
      receive(s2)
    ]
    end
```

```
fun simpleConvolution (i n: int) (x0: float) (write w: vector float)
  : vector float
  = begin
      if n = i then result.emit(write),
      a = [w0,w1,w2] = w.get(0,3),
      b = [x0,x1,x2] = [ x0 | write.get(i,2) ],
      write.set(i, Dotp(a,b)),
      simpleConvolution((i + 1),n,x1,write,w)
    end
```

ΠΣ

```
def zero : (T → T) → T → T := λ (s: T → T) (z: T), z
def succ : ((T → T) → T → T) → ((T → T) → T → T)
 := λ (w: (T → T) → T → T) (y: T → T) (x: T), y (w y x)
```

Π

```
def N := Π (A : U), (A → A) → A → A
def zero : N := λ (A : U) (S : A → A) (Z : A), Z
def succ : N -> N := λ (n : N) (A : U) (S : A → A) (Z : A), S (n A S Z)
def plus (m n : N) : N := λ (A : U) (S : A → A) (Z : A), m A S (n A S Z)
def mult (m n : N) : N := λ (A : U) (S : A → A) (Z : A), m A (n A S) Z
def pow (m n : N) : N := λ (A : U) (S : A → A) (Z : A), n (A → A) (m A) S Z
```

ΠΣ

```
def empty      : U := inductive { }
def L¹ (A : U) : U := inductive { nil | cons (head: A) (tail: L¹ A) }
def S¹         : U := inductive { base | loop : Equ S¹ base base }

def quot (A: U) (R : A -> A -> U) : U
 := inductive { quotient (a: A)
              | identification (a b: A) (r: R a b)
              : Equ (quot A R) (quotient a) (quotient b)
              }
```

```
def idfun  (A : U) : A → A := (λ (a : A), a
def idfun' (A : U) : A → A := transp (<i> A) 0
def idfun'' (A : U) : A → A := (λ (a : A), hcomp A 0 (λ (i : I), []) a
def isFiberBundle (B: U) (p: B → U) (F: U): U
 := Σ (v: U) (w: surjective v B), (Π (x: v), PathP (<_>U) (p (w.1 x)) F)
def ~~ (X : U) (a x' : X) : U := Path (ℑ X) (ι X a) (ι X x')
def 𝔻 (X : U) (a : X) : U := Σ (x' : X), ~~ X a x'
def unitDisc (X : U) (x : ℑ X) : U := Σ (x' : X), Path (ℑ X) x (ι X x')
def starDisc (X : U) (x : X) : 𝔻 X x := (x, idp (ℑ X) (ι X x))
def T^∞ (A : U) : U := Σ (a : A), 𝔻 A a
def inf-prox-ap (X Y : U) (f : X → Y) (x x' : X) (p : ~~ X x x')
  : ~~ Y (f x) (f x') := <i> ℑ-app X Y f (p @ i)
def d (X Y : U) (f : X → Y) (x : X) (ϵ : 𝔻 X x) : 𝔻 Y (f x)
 := (f ϵ.1, inf-prox-ap X Y f x ϵ.1 ϵ.2)
def T^∞-map (X Y : U) (f : X → Y) (τ : T^∞ X) : T^∞ Y
 := (f τ.1, d X Y f τ.1 τ.2)
def is-homogeneous (A : U)
 := Σ (e : A) (t : A → equiv A A),
    Π (x : A), Path A ((t x).1 e) x
```

O_{CPS_λ}

```
def CPS_λ : U
 := inductive { var (x: nat)
              | lam (l: nat) (d: cps)
              | app (f a: cps)
              }
```

O_{CPS}

```
objdump ./target/release/o -d | grep mulpd
   223f1: c5 f5 59 0c d3    vmulpd (%rbx,%rdx,8),%ymm1,%ymm1
   223f6: c5 dd 59 64 d3 20 vmulpd 0x20(%rbx,%rdx,8),%ymm4,%ymm4
   22416: c5 f5 59 4c d3 40 vmulpd 0x40(%rbx,%rdx,8),%ymm1,%ymm1
   2241c: c5 dd 59 64 d3 60 vmulpd 0x60(%rbx,%rdx,8),%ymm4,%ymm4
   2264d: c5 f5 59 0c d3    vmulpd (%rbx,%rdx,8),%ymm1,%ymm1
   22652: c5 e5 59 5c d3 20 vmulpd 0x20(%rbx,%rdx,8),%ymm3,%ymm3
```

O_{CPS}

```
def Lazy : U
 := inductive { Defer (otree: NodeId) (a: AST) (cont: Cont)
              | Continuation (otree: NodeId) (a: AST) (cont: Cont)
              | Return (a: AST)
              | Start
              }
```

```
def Cont : U
 := inductive { Expressions (a: AST) (v: Option (Iter AST)) (c: Cont)
              | Assign (ast: AST) (cont: Cont)
              | Cond (c,d: AST) (cont: Cont)
              | Func (a,b,c: AST) (cont: Cont)
              | List (acc: Vec AST) (vec: Iter AST) (i: Nat) (c: Cont)
              | Call (a: AST) (i: Nat) (cont: Cont)
              | Return
              | Intercore (m: Message) (cont: Cont)
              | Yield (cont: Cont)
              }
```

O_{CPS}

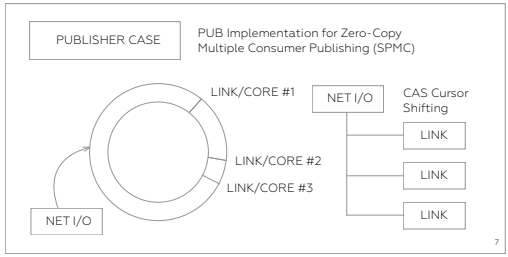
```
E: V | A | C
NC: ";" = [] | ";" m:NL = m
FC: ";" = [] | ";" m:FL = m
EC: ";" = [] | ";" m:EL = m
NL: NAME | o:NAME m:NC = Cons o m
FL: E | o:E | m:FC = Cons o m
EL: E | EC  | o:E m:EC = Cons o m
C: N | c:N a:C = Call c a
N: NAME | S | HEX | L | F
L: "(" ")" = [] | "([" c:NL "]" m:FL ")" = Table c m
                | "(" l:EL ")" = List l
F: "{" "}" = Lambda [] [] []
           | "{[" c:NL "]" m:EL "}" = Lambda [] c m
           | "{" m:EL "}" = Lambda [] [] m
```

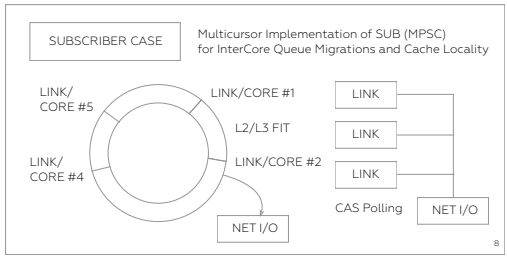
```
def AST : U
 := inductive { Atom (a: Scalar)
              | Vector (a: Vec AST)
              }

def Value : U
 := inductive { Nil
              | SymbolInt (a: u16)
              | SequenceInt (a: u16)
              | Number (a: i64)
              | Float (a: f64)
              | VecNumber (Vec i64)
              | VecFloat (Vec f64)
              }
```

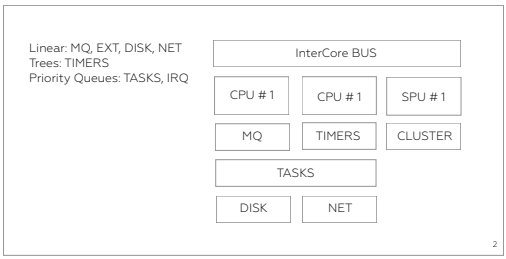
```
def Scalar : U
 := inductive { Nil
              | Any
              | List (a: AST)
              | Dict (a: AST)
              | Call (a b: AST)
              | Assign (a b: AST)
              | Cond (a b c: AST)
              | Lambda (otree: Option NodeId) (a b: AST)
              | Yield (c: Context)
              | Value (v: Value)
              | Name (s: String)
             }
```


https://github.com/voxoz/kernel

```
reactor[aux;0;mod[console;network]];
reactor[timercore;1;mod[timer]];
reactor[core1;2;mod[task]];
reactor[core2;3;mod[task]];
```


PUBLISHER CASE
PUB Implementation for Zero-Copy
Multiple Consumer Publishing (SPMC)
LINK/CORE #1
LINK/CORE #2
LINK/CORE #3
NET I/O
NET I/O
CAS Cursor
Shifting
LINK
LINK
LINK
7


SUBSCRIBER CASE
Multicursor Implementation of SUB (MPSC)
for InterCore Queue Migrations and Cache Locality
LINK/
CORE #5
LINK/
CORE #4
LINK/CORE #1
L2/L3 FIT
LINK/CORE #2
NET I/O
LINK
LINK
LINK
CAS Polling
NET I/O
8




```
fun pub(capacity: int): int
```

```
fun sub(publisher: int): int
```

```
fun spawn(core: int, program: code, cursors: array int): int
```

```
fun kill(process: int): int
```

```
fun send(writer: int, data: binary): int
```

```
fun receive(reader: int)
```

O_{CPS}

```rust
pub struct Publisher<T> {
    ring: Arc<RingBuffer<T>>,
    next: Cell<Sequence>,
    cursors: UncheckedUnsafeArc<Vec<Cursor>>,
}
```

```rust
pub struct Subscriber<T> {
    ring: Arc<RingBuffer<T>>,
    token: usize,
    next: Cell<Sequence>,
    cursors: UncheckedUnsafeArc<Vec<Cursor>>,
}
```

```rust
pub struct Channel {
    publisher: Publisher<Message>,
    subscribers: Vec<Subscriber<Message>>,
}
```

```rust
pub struct Memory<'a> {
    publishers: Vec<Publisher<Value<'a>>,
    subscribers: Vec<Subscriber<Value<'a>>>,
}
```

```rust
pub struct Scheduler<'a> {
    pub tasks: Vec<T3<Job<'a>>>,
    pub bus: Channel,
    pub queues: Memory<'a>,
    pub io: IO,
}
```

```
pub enum Message {
    Pub(Pub),
    Sub(Sub),
    Print(String),
    Spawn(Spawn),
    AckSub(AckSub),
    AckPub(AckPub),
    AckSpawn(AckSpawn),
    Exec(usize, String),
    Select(String, u16),
    QoS(u8, u8, u8),
    Halt,
    Nop,
}
```

```
def AVX-512 : U
 := inductive { Star | True | False
              | Variable (_: Var)
              | Prim (_: Builtin)
              | Int (_: nat) | Float (_: float)
              | Lambda (a: Var) (b: Linear) (c: Exp)
              | App (a b: Exp)
              | Pair (a b: Var) (c d: Exp)
              | Consume (a: Var) (b c: Exp)
              | Gen (a: Var) (b: Exp)
              | Spec (a: Exp) (b: Fraction)
              | Fix (a b: Var) (c d: Linear) (e: Exp)
              | If (a b c: Exp)
              | Let (a: Var) (b c: Exp)
              }
```

ω

1.
2.
3.
4.

---

https://github.com/o1
https://github.com/o3
https://ws.erp.uno
https://github.com/o89

5

6

$^{7}\mathsf{F}_{\omega}$

https://github.com/synrc

https://github.com/synrc

https://o89.github.io/lean4.dev/

```
axiom all : IO (List TableId)
axiom browse : IO String
axiom delete : Π (k: U), TableId -> k -> IO Unit
axiom first : Π (t: TableId) (k: U), IO (Maybe k)
axiom last : Π (k: U), TableId -> IO (Maybe k)
axiom foldl : Π (v acc: U), (v -> acc -> acc) -> acc -> TableId -> IO acc
axiom foldr : Π (v acc: U), (v -> acc -> acc) -> acc -> TableId -> IO acc
axiom insert : Π (v: U), TableId -> v -> IO Boolean
axiom lookup : Π (k v: U), TableId -> k -> IO (List v)
axiom member : Π (k: U), TableId -> k -> IO Boolean
axiom new : Atom -> TableOptions -> IO TableId
axiom next : Π (k: U), TableId -> k -> IO (Maybe k)
axiom prev : Π (k: U), TableId -> k -> IO (Maybe k)
axiom rename : TableId -> Atom -> IO Atom
axiom take : Π (k v: U), TableId -> k -> IO (List v)
axiom match : Π (a v: U), TableId -> a -> IO (List v)
axiom slot : Π (v: U), TableId -> Integer -> IO (Maybe (List v))
```

```
axiom pickle : Binary -> Binary
axiom depickle : Binary -> Binary
axiom encode : Π (k: U), k -> Binary
axiom decode : Π (k: U), Binary -> IO k
axiom reg: Π (k: U), k -> IO k
axiom unreg : Π (k: U), k -> IO k
axiom send : Π (k v z: U), k -> v -> IO z
axiom getSession : Π (k v: U), k -> IO v
axiom putSession : Π (k v: U), k -> v -> IO v
axiom getCache : Π (k v: U), Atom -> k -> IO v
axiom putCache : Π (k v: U), Atom -> k -> v -> IO v



data PI  = PI String Atom Atom Atom Integer RestartType
data Sup = Ok Pid String | Error String

axiom start : PI -> IO Sup
```

```
axiom get : Π (f k v: U), f -> k -> IO (Maybe v)
axiom put : Π (r: U), r -> IO StoreResult
axiom delete : Π (f k: U), f -> k -> StoreResult
axiom index : Π (f p v r: U), f -> Atom -> v -> List r

data Reader = Reader Integer Binary ETF String Integer
data Writer = Writer Integer Binary ETF String Integer
data StoreResult = Ok Integer String Binary
                 | Error Integer String Binary

axiom next : Reader -> IO Reader
axiom prev : Reader -> IO Reader
axiom take : Reader -> IO Reader
axiom drop : Reader -> IO Reader
axiom save : Reader -> IO Reader
axiom append : Π (f r: U), f -> r -> IO StoreResult
axiom remove : Π (f r: U), f -> r -> IO StoreResult
```

```
axiom start : Proc -> IO Sup
axiom stop : String -> IO Sup
axiom next : ProcId -> IO ProcRes
axiom load : ProcId -> IO ProcRes
axiom proc : ProcId -> IO ProcRes
axiom assign : ProcId -> IO ProcRes
axiom persist : ProcId -> Proc -> IO ProcRes
axiom amend : Π (k: U), ProcId -> k -> IO ProcRes
axiom discard : Π (k: U), ProcId -> k -> IO ProcRes
axiom modify : Π (k: U), ProcId -> k -> Atom -> IO ProcRes
axiom event : ProcId -> String -> IO ProcRes
axiom head : ProcId -> IO Hist
axiom hist : ProcId -> IO (List Hist)
axiom step : ProcId -> Atom -> IO (List Hist)
axiom docs : ProcId -> IO (List Tuple)
axiom events : ProcId -> IO (List Tuple)
axiom tasks : ProcId -> IO (List Tuple)
axiom doc : Tuple -> ProcId -> IO (List Tuple)

data ProcId = String
data Proc = Proc ProcId String
data ProcRes = Ok Integer String Binary
             | Error Integer String Binary
```

```
axiom q : Π (k: U), Atom -> k
axiom qc : Π (k: U), Atom -> k
```

```
axiom jse : Maybe Binary -> Binary
axiom hte : Maybe Binary -> Binary
axiom wire (actions: List Action) : IO (List Action)
axiom render (content: Either Action Element) : Binary
axiom insert_top (dom: Atom) (content: List Element) : IO (List Action)
axiom insert_bottom (dom: Atom) (content: List Element) : IO (List Action)
axiom update (dom: Atom) (content: List Element) : IO (List Action)
axiom clear (dom: Atom) : IO Unit
axiom remove (dom: Atom) : IO Unit
```

```
#textbox { id=userName, body= <<"Anonymous">> },
#panel { id=chatHistory, class=chat_history }
```

```
<input value="Anonymous" id="userName" type="text"/>
<div id="chatHistory" class="chat_history"></div>
```

```
nitro:update(loginButton,
    #button{id = loginButton,
        body = "Login",
        postback = login,
        source = [user, pass]});
```

```
qi('loginButton').outerHTML='<button id=\"loginButton\"
  type=\"button\">Login</button>'; { var x=qi('loginButton');
  x && x.addEventListener('click',function (event){
    event.preventDefault(); { if (validateSources(['user','pass'])) {
    ws.send(enc(tuple(atom('pickle'),bin('loginButton'),
      bin('b840bca20b3295619d1157105e355880f850bf0223f2f081549dc
      8934ecbcd3653f617bd96cc9b36b2e7a19d2d47fb8f9fbe32d3c768866
      cb9d6d85700416edf47b9b90742b0632c750a4240a62dfc56789e0f5d8
      590f9afdfb31f35fbab1563ec54fcb17a8e3bad463218d6402f1304'),
      [tuple(tuple(string('loginButton'),bin('detail')),[]),
      tuple(atom('user'),querySource('user')),
      tuple(atom('pass'),querySource('pass'))])); }
  else console.log('Validation Error'); }});};
```


https://n2o.dev/ua/deps/nitro/index.html

[1][2][3]$\Pi\Sigma^{\infty}$

PTSCoCHenkAlonzo[4]

ω

Morte[5] cubicaltt[6]

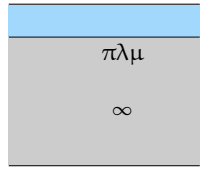
$$\begin{cases} \mathrm{Sorts} = \mathcal{U}.\{i\}, i : \mathrm{Nat} \\ \mathrm{Axioms} = \mathcal{U}.\{i\} : \mathcal{U}.\{\mathrm{inci}\} \\ \mathrm{Rules} = \mathcal{U}.\{i\} \rightsquigarrow \mathcal{U}.\{j\} : \mathcal{U}.\{\mathrm{maxij}\} \end{cases}$$

HenkHenk

HenkΠλβηHenk

---

https://github.com/Gabriel439/Haskell-Morte-Library
https://iohk.io/research/papers/#JT5XKNBP



7

ω
ω
ω

```
I := list nat
U := U + U_nat
O := U + I + ( O ) + O O + O → O
   + λ ( I : O ) → O
   + ∀ ( I : O ) → O
```

$\rightarrow \lambda \forall O_{PTS} O_{\Pi}$

```
def PTS (lang: U) : U
 := inductive { 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)
              }
```

$\infty_{\omega^{8}}$ Fixpoint

$$U_0 : U_1 : U_2 : U_3 : \ldots$$

$U_0 U_1 U_2 U_3$

$$\frac{o : Nat}{U_o}$$

$$\frac{}{U : U}$$

A_1

$$\frac{i : Nat, j : Nat, i < j}{U_i : U_j}$$

R_1

$$\frac{i : Nat, j : Nat}{U_i \to U_j : U_{max(i,j)}}$$

A_2

$$\frac{i : Nat}{U_i : U_{i+1}}$$

R_2

$$\frac{i : Nat, j : Nat}{U_i \to U_j : U_j}$$

list Sigma

$$\frac{}{\Gamma : Ctx}$$

1

$$\frac{\Gamma : Ctx}{\emptyset : \Gamma}$$

2

$$\frac{A : U_i, x : A, \Gamma : Ctx}{(x : A) \vdash \Gamma : Ctx}$$

$\beta\eta O_{PTS} \beta\eta$

Π

$$\frac{A : U_i, x : A \vdash B : U_j}{\Pi(x : A) \to B : U_{p(i,j)}}$$

λ

$$\frac{x : A \vdash b : B}{\lambda(x : A) \to b : \Pi(x : A) \to B}$$

App

$$\frac{f : (\Pi(x : A) \to B) a : A}{fa : B[a/x]}$$

β

$$\frac{x : A \vdash b : B a : A}{(\lambda(x : A) \to b)a = b[a/x] : B[a/x]}$$

$$\frac{\pi_1 : A u : A \vdash \pi_2 : B}{[\pi_1/u]\pi_2 : B}$$

$O_= O_I$

```
remoteremote
```

```
type (:star,N)     D ⟶ (:star,N+1)
     (:var,N,I)    D ⟶ :true = proplists:is_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)
```

```
  sh (:star,X)     N P ⟶ (:star,X)
     (:var,N,I)    N P ⟶ (:var,N,I+1) when I >= P
                       ⟶ (:var,N,I)
     (:pi,N,O,I,O) N P ⟶ (:pi,N,O,sh I N P,sh O N P+1)
     (:fn,N,O,I,O) N P ⟶ (:fn,N,O,sh I N P,sh O N P+1)
     (:app,L,R)    N P ⟶ (:app,L,R)
```

```
sub (:star,X)     N V L → (:star,X)
    (:var,N,L)    N V L → V
    (:var,N,I)    N V L → (:var,N,I-1) when I > L
    (:pi,N,O,I,O) N V L → (:pi,N,O,sub I N V L,sub O N (sh V N O) L+1)
    (:pi,F,X,I,O) N V L → (:pi,F,X,sub I N V L,sub O N (sh V F O) L)
    (:fn,N,O,I,O) N V L → (:fn,N,O,sub I N V L,sub O N (sh V N O) L+1)
    (:fn,F,X,I,O) N V L → (:fn,F,X,sub I N V L,sub O N (sh V F O) L)
    (:app,F,A)    N V L → (:app,   sub F N V L,sub A N V L)


norm (:star,X)     → (:star,X)
    (:var,X)       → (:var,X)
    (:pi,N,O,I,O)  → (:pi,N,O,norm I,norm O)
    (:fn,N,O,I,O)  → (:fn,N,O,norm I,norm O)
    (:app,F,A)     → case norm F of
                         (:fn,N,O,I,O) → norm (subst O N A)
                                     NF → (:app,NF,norm A) end


  eq (:star,N)        (:star,N)        → true
     (:var,N,I)       (:var,(N,I))     → true
     (:pi,N1,O,I1,O1) (:pi,N2,O,I2,O2) →
          let :true = eq I1 I2
           in eq O1 (subst (shift O2 N1 0) N2 (:var,N1,O) O)
     (:fn,N1,O,I1,O1) (:fn,N2,O,I2,O2) →
          let :true = eq I1 I2
           in eq O1 (subst (shift O2 N1 0) 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))
```

```
> ./om help me
[{a,[expr],"to parse. Returns {_,_} or {error,_}."},
 {type,[term],"typechecks and returns type."},
 {erase,[term],"to untyped term. Returns {_,_}."},
 {norm,[term],"normalize term. Returns term's normal form."},
 {file,[name],"load file as binary."},
 {str,[binary],"lexical tokenizer."},
 {parse,[tokens],"parse given tokens into {_,_} term."},
 {fst,[{x,y}],"returns first element of a pair."},
 {snd,[{x,y}],"returns second element of a pair."},
 {debug,[bool],"enable/disable debug output."},
 {mode,[name],"select metaverse folder."},
 {modes,[],"list all metaverses."}]

> ./om print fst erase norm a "#List/Cons"
   \ Head
-> \ Tail
-> \ Cons
-> \ Nil
-> Cons Head (Tail Cons Nil)
ok
```

$(\mu L_A, in) L_A(X) = 1 + (A \times X) \mu L_A = List(A) nil : 1 \to List(A) cons : A \times List(A) \to List(A) nil = in \circ inl cons = in \circ inr in = [nil, cons]$
$c : 1 \to C h : A \times C \to C f = (\!|[c, h]|\!) : List(A) \to C$

$$\begin{cases} f \circ nil = c \\ f \circ cons = h \circ (id \times f) \end{cases}$$

$f = foldr(c, h) \mu(1 + A \times X)[1 \to List(A), A \times List(A) \to List(A)]$
OPTS

$$\begin{cases} foldr = (\!|[f \circ nil, h]|\!), f \circ cons = h \circ (id \times f) \\ len = (\!|[zero, \lambda an \to succn]|\!) \\ (++) = \lambda xsys \to (\!|[\lambda(x) \to ys, cons]|\!)(xs) \\ map = \lambda f \to (\!|[nil, cons \circ (f \times id)]|\!) \end{cases}$$

```
def list (A: U) : U
 := inductive { cons (x: A) (cs: list A)
              | nil
              }
```

$$\begin{cases} list = \lambda ctor \to \lambda cons \to \lambda nil \to ctor \\ cons = \lambda x \to \lambda xs \to \lambda list \to \lambda cons \to \lambda nil \to consx(xslistconsnil) \\ nil = \lambda list \to \lambda cons \to \lambda nil \to nil \end{cases}$$

```
axiom map (A B: U) (f: A -> B) : list A -> list B
axiom length (A: U): list A -> nat
axiom append (A: U): list A -> list A -> list A
axiom foldl (A B: U) (f: B -> A -> B) (Z: B): list A -> B
axiom filter (A: U) (p: A -> bool) : list A -> list A
```

$$\begin{cases} len = foldr(\lambda xn \to succn)0 \\ (++) = \lambda ys \to foldrconsys \\ map = \lambda f \to foldr(\lambda xxs \to cons(fx)xs)nil \\ filter = \lambda p \to foldr(\lambda xxs \to ifpxthenconsxxselsexs)nil \\ foldl = \lambda fvxs = foldr(\lambda xg \to (\lambda \to g(fax)))idxsv \end{cases}$$

$F_A(X) = 1 + A \times XF_A(X) = A + X \times X1AF_A(X) = A \times X$

$\mu X \to 1 + X$
$\mu X \to 1 + A \times X$
$\mu X \to 1 + X \times X + X$
$\nu X \to A \times X$
$\nu X \to 1 + A \times X$
$\mu X \to \mu Y \to 1 + X \times Y = \mu X = \mathrm{List}X$

$\Pi\Pi$FixpointfunExthomotopy

Henkλλ

```
ext (:var,X,N,F)      → (:var,X)
    (:app,A,B,N,F)    → (:call,N,ext(F,A,N),[ext(F,B,N)])
    (:fn,S,_,I,O,N,F) → (:fun,N,(:clauses,[{:clause,N,
                                  [(:var,N,S)],[],[ext(F,O,N)]}]))
                    _ → []
```

∞Henk

HenkOPTSHenkHenk

$W012$

W
$$\frac{A : Type \quad x : A \quad B(x) : Type}{W(x : A) \to B(x) : Type}$$

W
$$\frac{a : A \quad t : B(a) \to W}{sup(a, t) : W}$$

W
$$\frac{\begin{array}{l} w : W \vdash C(w) : Type \\ x : A, u : B(x) \to W, \\ v : \Pi(y : B(x)) \to C(u(y)) \vdash c(x, u, v) : C(sup(x, u)) \end{array}}{w : W \vdash wrec(w, c) : C(w)}$$

$W\beta$
$$\frac{\begin{array}{l} w : W \vdash C(w) : \\ x : A, u : B(x) \to W, \\ v : \Pi(y : B(x)) \to C(u(y)) \vdash c(x, u, v) : C(sup(x, u)) \end{array}}{\begin{array}{l} x : A, u : B(x) \to W \vdash wrec(sup(x, u), c) \\ \quad = c(x, u, \lambda(y : B(x)), wrec(u(y), c)) : C(sup(x, u)) \end{array}}$$

WMLTTPer

CICFrankChristine

```
file -> 'module' id 'do' content : {module,'2','4'} .
content -> 'empty' : [] .
content -> line content : ['1'|'2'] .
line -> 'import' id : {import,'2'} .
line -> 'def' id tele ':' exp ':=' exp : {def,'2','5','3','7'} .
tele -> 'empty' : [] .
tele -> optic tele: ['1'|'2'] .
optic -> '(' vars ':' exp ')' : {'()','2','4'} .
vars -> id : '1' .
vars -> id vars : ['1'|'2'] .
exp -> id : '1' .
exp -> id '.' exp : {'.','1','3'} .
exp -> exp exp : {app,'1','2'} .
exp -> '(' exp ')' : '2' .
exp -> forall tele ',' exp : {'Π','2','4'} .
exp -> sigma tele ',' exp : {'Σ','2','4'} .
exp -> exp arrow exp : →{'','1','3'} .
exp -> lam tele ',' exp : {'λ','2','4'} .
exp -> '?' : {hole} .
exp -> U : {'U'} .
exp -> app : '1' .
exp -> 'inductive' '{' sum '}' : {inductive,lists:flatten(uncons('3'))} .
constructor -> id tele : {element,'1','2'} .
constructor -> id tele ':' exp : {interval,'1','2','4'} .
sum -> 'empty' : [] .
sum -> constructor : '1' .
sum -> constructor '|' sum : ['3'|'1'] .
app -> exp exp : {app,'1','2'} .
```

$$I : \square_n^{op} \to \mathrm{Set}\mathrm{CCHMHTSAnders}$$

```
start <Module.file> file
start <Module.command> repl
repl : COLON IDENT exp1 EOF | COLON IDENT EOF | exp0 EOF | EOF
file : MODULE IDENT WHERE line* EOF
path : IDENT
line : IMPORT path+ | OPTION IDENT IDENT | declarations
ident : IRREF | IDENT
lense : LPARENS ident+ COLON exp1 RPARENS
telescope : lense telescope
params : telescope | []
declarations :
  | DEF IDENT params DEFEQ exp1
  | DEF IDENT params COLON exp1 DEFEQ exp1
  | AXIOM IDENT params COLON exp1
face : LPARENS IDENT IDENT IDENT RPARENS
part : face+ ARROW exp1
exp0 : exp1 COMMA exp0 | exp1
exp1 : LSQ separated(COMMA, part) RSQ
  | LAM telescope COMMA exp1    | PI telescope COMMA exp1
  | SIGMA telescope COMMA exp1  | LSQ IRREF ARROW exp1 RSQ
  | LT ident+ GT exp1           | exp2 ARROW exp1
  | exp2 PROD exp1              | exp2
exp2 : exp2 exp3 | exp2 APPFORMULA exp3 | exp3
exp3 : LPARENS exp0 RPARENS LSQ exp0 MAP exp0 RSQ
  | HOLE               | PRE           | KAN           | IDJ exp3
  | exp3 FST           | exp3 SND      | NEGATE exp3   | INC exp3
  | exp3 AND exp3      | exp3 OR exp3  | ID exp3       | REF exp3
  | OUC exp3           | PATHP exp3    | PARTIAL exp3  | IDENT
  | IDENT LSQ exp0 MAP exp0 RSQ        | HCOMP exp3
  | LPARENS exp0 RPARENS               | TRANSP exp3 exp3
```

$|\langle\rangle\backslash\backslash\to\mathbf{moduleimportdataspli}\mathit{twherecompfillGlueglue}\mathbf{unglue}.1.2,$
$\mathit{cubical}$

$$\mathrm{Per}\mathcal{U}_i\Pi\tau = \mathbf{type}\mathrm{PerAnders}\Sigma\tau\Pi$$

$$\mathbb{N}\omega$$

$$\mathbb{N}\mathbf{Anders}\omega = \{\mathbf{V}_i, \mathbf{U}_i\}$$

$$=\alpha\beta$$

$$=V_i\mathcal{U}_i\Pi$$

τ

$\tau = \{\mathbf{infer}, \mathbf{app}, \mathbf{check}, \mathbf{act}, \mathbf{conv}, \mathbf{eval}\}$

Σ

Σ

$\mathcal{P}$

$\int = \{\Pi, \Sigma, =, \mathbf{W}, 0, 1, 2, \mathbf{Path}, \mathbf{Glue}\}$

$e_i : O_{n+1} \to O_n O_{CPS} = O_0$
O_{PTS}[9] O_{CCHM}[10]

https://github.com/groupoid/henk
https://github.com/groupoid/anders

$\Pi\Sigma=$

123

$^{4}\infty$

$\bot$

$\top$

$A \vee B$

$A \wedge B$

$A \Rightarrow B$

$x)$ $\exists_{x:A} B(x)$

$x)$ $\forall_{x:A} B(x)$

$=_A$ A^I

PerAndersAndersAnders

Anders

AndersPer

Anders

ΠΣId

ΠΣ

ΠΣ

Π

ΠΠ

$$\Pi\Pi f : \Pi(x : A), B(x) A B : A \to U_i$$

$$\Pi : U =_{def} \prod_{x:A} B(x).$$

```
def Pi (A: U) (B: A → U): U := Π (x: A), B(x)
```

$$\Pi \lambda x.b(x) x \mapsto b(x)$$

$$\backslash(x : A) \to b(x) =_{def} \prod_{A:U} \prod_{B:A\to U} \prod_{b:\prod_{a:A} B(a)} \lambda x.b(x) : \prod_{y:A} B(y).$$

```
def lambda (A: U) (B: A → U) (b: Pi A B) : Pi A B := λ (x : A), b x
def lam (A B: U) (f: A → B) : A → B :=  λ (x : A), f x
```

$$f : A \to BP_\omega$$

ΠΠΠλ

$$fa =_{def} \prod_{A:U} \prod_{B:A\to U} \prod_{a:A} \prod_{f:\prod_{x:A} B(a)} f(a) : B(a).$$

```
def apply (A B: U) (f: A → B) (a: A) : B := f a
def app (A: U) (B: A → U) (a: A) (f: Pi A B): B a := f a
```

```
def ∘ᵀ (a b c: U) : U := (b → c) → (a → b) → (a → c)
def ∘ (a b c : U) : ∘ᵀ a b c := λ (g: b → c) (f: a → b) (x: a), g (f x)
```

Πβlam ∘ app

$$f(a) =_{B(a)} (\lambda(x : A) \to f(a))(a).$$

```
def Π-β (A : U) (B : A → U) (a : A) (f : Pi A B)
  : Path (B a) (Π-apply A B (Π-lambda A B f) a) (f a)
 := idp (B a) (f a)
```

Πηapp ∘ lam

$$f =_{(x:A)\to B(a)} (\lambda(y : A) \to f(y)).$$

```
def Π-η (A : U) (B : A → U) (a : A) (f : Pi A B)
  : Path (Pi A B) f (λ (x : A), f x)
 := idp (Pi A B) f
```

ΠΠ

$$p : E \to B y : B x : E p(x) = y$$

$$F \to E \xrightarrow{p} B E F B(F, E, p, B) p : E \to B y : B U_b f : p^{-1}(U_b) \to U_b \times F$$

$$p^{-1}(U_b) \xrightarrow{f} U_b \times F$$
$$U_b \xleftarrow{pr_1}$$

$$FBapp : F \times B \to E$$
$$F \times B \xrightarrow{app} E \xrightarrow{pr_1} B$$

$$pr_1 pr_1 appSet_{/B} FAA \times B \to ESet_{/B} A \to F$$

$$E\Sigma(B, F)p = pr_1(F, \Sigma(B, F), pr_1, B)$$

$$(F, B * F, pr_1, B)y : BF(y)$$

```
FiberPi (B: U) (F: B -> U) (y: B)
      : Path U (fiber (Sigma B F) B (pi1 B F) y) (F y)
```

ΠΣ

$$g : B \to AC\Pi_g : C_{/B} \to C_{/A}$$

$$\mathbf{H}(\infty, 1)E \to B : \mathbf{H}_{/B} \mathbf{H}\Gamma_\Sigma(E)$$

$$\Gamma_\Sigma(E) = \Pi_\Sigma(E) \in \mathbf{H}.$$

```
setFun (A B : U) (_: isSet B) : isSet (A -> B)
```

```
piIsContr (A: U) (B: A -> U) (u: isContr A)
          (q: (x: A) -> isContr (B x)) : isContr (Pi A B)
```

$$f : A \to Bg : B \to Af \circ g : B \xrightarrow{g} A \xrightarrow{f} BB$$

Σ

$\Sigma\Sigma$

$\Sigma\Sigma AB\mathcal{U}A : \mathcal{U}B : A \to \mathcal{U}a : AB(a)\Sigma(A,B)(x,B(x))$

```
def Sigma (A : U) (B : A -> U) : U := Σ (x : A), B x
```

$\Sigma(a,b)\Sigma(A,B)\Sigma(A,B)(a,b)b : B(a)$

```
def dpair (A: U) (B: A -> U) (a: A) (b: B a) : Sigma A B = (a,b)
```

$\Sigma\Sigma$

```
def pr1 (A: U) (B: A -> U) (x: Sigma A B): A := x.1
def pr2 (A: U) (B: A -> U) (x: Sigma A B): B (pr1 A B x) := x.2
def sigInd (A: U) (B: A -> U) (C: Sigma A B -> U)
    (g: (a: A) (b: B a) -> C (a, b)) (p: Sigma A B)
  : C p := g p.1 p.2
```

$\Sigma pr_1 pr_2(a,b)abEqu$

```
def Beta1 (A: U) (B: A -> U) (a:A) (b: B a)
  : Equ A a (pr1 A B (a,b))

def Beta2 (A: U) (B: A -> U) (a: A) (b: B a)
  : Equ (B a) b (pr2 A B (a,b))
```

$\Sigma p : \Sigma(A,B)(pr_1, pr_2)$

```
def Eta2 (A: U) (B: A -> U) (p: Sigma A B)
  : Equ (Sigma A B) p (pr1 A B p,pr2 A B p)
```

$f : A \to BC\Sigma_f : C_{/A} \to C_{/B}$

$x : Ay : BR(x,y)f : A \to Bx : AR(x,f(x))$

```
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))



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))
```

$\Sigma\Sigma$

```
setSig (A:U) (B: A -> U) (sA: isSet A)
       (sB : (x:A) -> isSet (B x)) : isSet (Sigma A B)
```

$$t, u : \Sigma(A, B)p : t_1 =_A u_1)(t_2 =_{B(p@i)} u_2)$$

```
pathSig (A:U) (B : A -> U) (t u : Sigma A B)
      : Path U (Path (Sigma A B) t u)
               ((p: Path A t.1 u.1) * PathP (<i>B(p@i)) t.2 u.2)
```

$[0,1]^{56789}$

```
Hetero (A B: U) (a: A) (b: B) (P: Path U A B) : U = PathP P a b
Path (A: U) (a b: A) : U = PathP (<i> A) a b
```

$[0,1]$<i>a$\lambda(i : I) \to a$

```
refl (A: U) (a: A) : Path A a a
```

```
app1 (A: U) (a b: A) (p: Path A a b): A = p @ 0
app2 (A: U) (a b: A) (p: Path A a b): A = p @ 1
```

$$\lambda(i : I) \to \overset{\text{comp}}{\overset{a}{\overset{a}{a}}}$$

```
composition (A: U) (a b c: A) (p: Path A a b) (q: Path A b c)
          : Path A a c = comp (<i>Path A a (q@i)) p []
```

```
inv (A: U) (a b: A) (p: Path A a b): Path A b a = <i> p @ -i
```

$\lambda(i, j : I) \to p@min(i, j)\lambda(i, j : I) \to p@max(i, j)$

$$\lambda(i : I)\overset{\lambda(i : I) \to a}{\overset{a}{\overset{a}{a}}} \to a \qquad \overset{\lambda(i : I) \to b}{\overset{p}{\overset{p}{p}}}\lambda(i : I) \to b$$

```
connection1 (A: U) (a b: A) (p: Path A a b)
          : PathP (<x> Path A (p@x) b) p (<i>b)
          = <y x> p @ (x \/ y)

connection2 (A: U) (a b: A) (p: Path A a b)
          : PathP (<x> Path A a (p@x)) (<i>a) p
          = <x y> p @ (x /\ y)
```

$[0,1]\lambda\to$

---

https://5ht.co/cubicaltt.pdf
https://5ht.co/cctt.pdf
http://www.cse.chalmers.se/~coquand/mod1.pdf

https://www.cs.cmu.edu/~cangiuli/papers/ccctt.pdf
https://arxiv.org/pdf/1712.04864.pdf

```
ap  (A B: U) (f: A -> B)
    (a b: A) (p: Path A a b)
  : Path B (f a) (f b)

apd (A: U) (a x:A) (B: A -> U) (f: A -> B a)
    (b: B a) (p: Path A a x)
  : Path (B a) (f a) (f x)
```

p

```
trans (A B: U) (p: Path U A B) (a: A) : B
```

```
singl (A: U) (a: A): U = (x: A) * Path A a x



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



contr (A: U) (a b: A) (p: Path A a b)
  : Path (singl A a) (eta A a) (b,p)
  = <i> (p @ i,<j> p @ i/\j)



D (A: U) : U = (x y: A) -> Path A x y -> U
J (A: U) (x y: A) (C: D A)
  (d: C x x (refl A x))
  (p: Path A x y) : C x y p
= subst (singl A x) T (eta A x) (y, p) (contr A x y p) d where
  T (z: singl A x) : U = C x (z.1) (z.2)



J (A: U) (a b: A)
  (P: singl A a -> U)
  (u: P (a,refl A a))
  (p: Path A a b) : P (b,p)



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



trans_comp (A: U) (a: A)
  : Path A a (trans A A (<_> A) a)
  = fill (<i> A) a []
subst_comp (A: U) (P: A -> U) (a: A) (e: P a)
  : Path (P a) e (subst A P a a (refl A a) e)
  = trans_comp (P a) e
J_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))
  = subst_comp (singl A a) T (eta A a) d where T (z: singl A a)
  : U = C a (z.1) (z.2)
```

$U_{n\in N}U_0$

$U_i : U_j, i, j \in Ni, j$

$U_i \to U_j : U_{\lambda(i,j),i,j\in N}\lambda : N \times N \to N$

$\lambda max : N \times N \to N$

$\lambda snd : N \times N \to N$

$U_i, i \in N\in$

$U_{n\in N}U_i : U_j, i, j \in NU_i \to U_j : U_{\lambda(i,j),i,j\in N}\lambda max snd$

$\{\{\star, \Box\}, \{\star : \Box\}, \{i \to j : j; i, j \in \{\star, \Box\}\}\star\Box\lambda = snd$

$PTS^\infty\{U_{i\in N}, U_i : U_{j;i<j;i,j\in N}, U_i \to U_j : U_{\lambda(i,j);i,j\in N}\}U_i iiPTS^\infty$[10]

$$\gamma_0 : \Gamma =_{def} \star.$$

$$\Gamma; A =_{def} \sum_{\gamma:\Gamma} A(\gamma).$$

$$\Gamma \vdash A =_{def} \prod_{\gamma:\Gamma} A(\gamma).$$

ΠΣ

ΠΣΣ

```
def MLTT (A: U) : U := Σ
    (Π-form  : Π (B: A → U), U)
    (Π-ctor₁ : Π (B: A → U), Pi A B → Pi A B)
    (Π-elim₁ : Π (B: A → U), Pi A B → Pi A B)
    (Π-comp₁ : Π (B: A → U) (a: A) (f: Pi A B),
               Equ (B a) (Π-elim₁ B (Π-ctor₁ B f) a) (f a))
    (Π-comp₂ : Π (B: A → U) (a:  A) (f: Pi A B),
               Equ (Pi A B) f (λ (x : A), f x))
    (Σ-form  : Π (B: A → U), U)
    (Σ-ctor₁ : Π (B: A → U) (a : A) (b : B a), Sigma A B)
    (Σ-elim₁ : Π (B: A → U) (p : Sigma A B), A)
    (Σ-elim₂ : Π (B: A → U) (p : Sigma A B), B (pr₁ A B p))
    (Σ-comp₁ : Π (B: A → U) (a : A) (b: B a),
               Equ A a (Σ-elim₁ B (Σ-ctor₁ B a b)))
    (Σ-comp₂ : Π (B: A → U) (a : A) (b: B a),
               Equ (B a) b (Σ-elim₂ B (a, b)))
    (Σ-comp₃ : Π (B: A → U) (p : Sigma A B),
               Equ (Sigma A B) p (pr₁ A B p, pr₂ A B p))
    (=-form  : Π (a: A), A → U)
    (=-ctor₁ : Π (a: A), Equ A a a)
    (=-elim₁ : Π (a: A) (C: D A) (d: C a a (=-ctor₁ a))
                 (y: A) (p: Equ A a y), C a y p)
    (=-comp₁ : Π (a: A) (C: D A) (d: C a a (=-ctor₁ a)),
               Equ (C a a (=-ctor₁ a)) d (=-elim₁ a C d a (=-ctor₁ a))), U



theorem instance (A : U) : MLTT A :=
    (Pi A, lambda A, app A, comp₁ A, comp₂ A,
     Sigma A, pair A, pr₁ A, pr₂ A, comp₃ A, comp₄ A, comp₅ A,
     Equ A, refl A, J A, comp₆ A, A)
```

mltt.cttcubicaltt

```
$ rlwrap ./anders.native check ./experiments/mltt.anders
File loaded.
> :n instance
TYPE: Π (A : U), Σ (Π-form : Π (B : (A → U)), U), Σ (Π-ctor₁ : Π (B : (A
      → U)), (Π (x : A), (B x) → Π (x : A), (B x))), Σ (Π-elim₁ : Π (B : (A
       → U)), (Π (x : A), (B x) → Π (x : A), (B x))), Σ (Π-comp₁ : Π (B : (
      A → U)), Π (a : A), Π (f : Π (x : A), (B x)), Π (P : ((B a) → U)), ((
      P (((Π-elim₁ B) ((Π-ctor₁ B) f)) a)) → (P (f a)))), Σ (Π-comp₂ : Π (
      B : (A → U)), Π (a : A), Π (f : Π (x : A), (B x)), Π (P : (Π (x : A),
       (B x) → U)), ((P f) → (P λ (x : A), (f x)))), Σ (Σ-form : Π (B : (A
      → U)), U), Σ (Σ-ctor₁ : Π (B : (A → U)), Π (a : A), Π (b : (B a)), Σ
       (x : A), (B x)), Σ (Σ-elim₁ : Π (B : (A → U)), Π (p : Σ (x : A), (B
      x)), A), Σ (Σ-elim₂ : Π (B : (A → U)), Π (p : Σ (x : A), (B x)), (B p
      .1)), Σ (Σ-comp₁ : Π (B : (A → U)), Π (a : A), Π (b : (B a)), Π (P :
       (A → U)), ((P a) → (P ((Σ-elim₁ B) (((Σ-ctor₁ B) a) b))))), Σ (Σ-comp
      ₂ : Π (B : (A → U)), Π (a : A), Π (b : (B a)), Π (P : ((B a) → U)), (
      (P b) → (P ((Σ-elim₂ B) (a, b))))), Σ (Σ-comp₃ : Π (B : (A → U)), Π (
      p : Σ (x : A), (B x)), Π (P : (Σ (x : A), (B x) → U)), ((P p) → (P (p
      .1, p.2)))), Σ (=-form : Π (a : A), (A → U)), Σ (=-ctor₁ : Π (a : A),
       Π (P : (A → U)), ((P a) → (P a))), Σ (=-elim₁ : Π (a : A), Π (C : Π
      (x : A), Π (y : A), (Π (P : (A → U)), ((P x) → (P y)) → U)), Π (d : ((
      (C a) a) (=-ctor₁ a))), Π (y : A), Π (p : Π (P : (A → U)), ((P a) → (P
       y))), (((C a) y) p)), Σ (=-comp₁ : Π (a : A), Π (C : Π (x : A), Π (y
       : A), (Π (P : (A → U)), ((P x) → (P y)) → U)), Π (d : ((((C a) a) (=-
      ctor₁ a))), Π (P : ((((C a) a) (=-ctor₁ a)) → U)), ((P d) → (P (((((=-
      elim₁ a) C) d) a) (=-ctor₁ a))))), U
NORMEVAL: λ (A : U), (λ (B : (A → U)), Π (x : A), (B x), (λ (B : (A → U)), λ
       (b : Π (x : A), (B x)), λ (x : A), (b x), (λ (B : (A → U)), λ (f : Π
      (x : A), (B x)), λ (a : A), (f a), (λ (B : (A → U)), λ (a : A), λ (f :
      Π (x : A), (B x)), λ (P : ((B a) → U)), λ (u : (P (f a))), u, (λ (B :
      (A → U)), λ (a : A), λ (f : Π (x : A), (B x)), λ (P : (Π (x : A), (B x
      ) → U)), λ (u : (P f)), u, (λ (B : (A → U)), Σ (x : A), (B x), (λ (B :
      (A → U)), λ (a : A), λ (b : (B a)), (a, b), (λ (B : (A → U)), λ (x : Σ
       (x : A), (B x)), x.1, (λ (B : (A → U)), λ (x : Σ (x : A), (B x)), x.2,
       (λ (B : (A → U)), λ (a : A), λ (b : (B a)), λ (P : (A → U)), λ (u : (P
       a)), u, (λ (B : (A → U)), λ (a : A), λ (b : (B a)), λ (P : ((B a) → U)
      ), λ (u : (P b)), u, (λ (B : (A → U)), λ (p : Σ (x : A), (B x)), λ (P
      : (Σ (x : A), (B x) → U)), λ (u : (P p)), u, (λ (x : A), λ (y : A), Π
      (P : (A → U)), ((P x) → (P y)), (λ (x : A), λ (P : (A → U)), λ (u : (P
      x)), u, ((J A), ((comp₆ A), A)))))))))))))))))
```

```
data empty =
emptyRec (C: U): empty -> C = split {}
emptyInd (C: empty -> U): (z: empty) -> C z = split {}

data unit = star
unitRec (C: U) (x: C): unit -> C = split tt -> x
unitInd (C: unit -> U) (x: C tt): (z: unit) -> C z = split tt -> x
```

```
data bool = false | true
b1: U = bool -> bool
b2: U = bool -> bool -> bool
negation: b1 = split { false -> true; true -> false }
or: b2 = split { false -> idfun bool; true -> lambda bool bool true }
and: b2 = split { false -> lambda bool bool false; true -> idfun boo }
boolEq: b2 = lamb bool (bool -> bool) negation
boolRec (C: U) (f t: C): bool -> C = split { false -> f ; true -> t }
boolInd (C: bool -> U) (f: A false) (t: A true): (n:bool) -> A n
  = split { false -> f ; true -> t }
```

$$M_A(X) = 1 + A$$

```
data maybe (A: U) = nothing | just (x: A)
maybeRec (A P: U) (n: P) (j: A -> P): maybe A -> P
       = split { nothing -> n; just a -> j a }

maybeInd (A: U) (P: maybe A -> U) (n: P nothing)
         (j: (a: A) -> P (just a)): (a: maybe A) -> P a
       = split { nothing -> n ; just x -> j x }



data either (A B: U) = left (x: A) | right (y: B)
eitherRec (A B C: U) (b: A -> C) (c: B -> C): either A B -> C
        = split { inl x -> b(x) ; inr y -> c(y) }

eitherInd (A B: U) (C: either A B -> U)
          (x: (a: A) -> C (inl a))
          (y: (b: B) -> C (inr b))
        : (x: either A B) -> C x
        = split { inl i -> x i ; inr j -> y j }



data tuple (A B: U) = pair (x: A) (y: B)
prod (A B: U) (x: A) (y: B): (_: A) * B = (x,y)
tupleRec  (A B C: U) (c: (x:A) (y:B) -> C): (x: tuple A B) -> C
        = split pair a b -> c a b
tupleInd  (A B: U) (C: tuple A B -> U)
          (c: (x:A)(y:B) -> C (pair x y))
        : (x: tuple A B) -> C x
        = split pair a b -> c a b
```

```
data nat = zero | succ (n: nat)
natEq: nat -> nat -> bool
natCase (C:U) (a b: C): nat -> C
natRec  (C:U) (z: C) (s: nat->C->C) : (n:nat) -> C


natElim (C:nat->U) (z: C zero)
        (s: (n:nat)->C(succ n)): (n:nat) -> C(n)
natInd  (C:nat->U) (z: C zero)
        (s: (n:nat)->C(n)->C(succ n)): (n:nat) -> C(n)
```

$(\mu L_A, \mathrm{in}) L_A(X) = 1 + (A \times X) \mu L_A = \mathrm{List}(A) \mathrm{nil} : 1 \to \mathrm{List}(A) \mathrm{cons} : A \times \mathrm{List}(A) \to \mathrm{List}(A) \mathrm{nil} = \mathrm{in} \circ \mathrm{inl} \mathrm{cons} = \mathrm{in} \circ \mathrm{inr} \mathrm{in} = [\mathrm{nil}, \mathrm{cons}]$

```
data list (A: U) = nil | cons (x:A) (xs: list A)
listCase (A C:U) (a b: C): list A -> C
listRec (A C:U) (z: C) (s: A->list A->C->C): (n:list A) -> C
listElim (A: U) (C:list A->U) (z: C nil)
   (s: (x:A)(xs:list A)->C(cons x xs)): (n:list A) -> C(n)
listInd (A: U) (C:list A->U) (z: C nil)
   (s: (x:A)(xs:list A)->C(xs)->C(cons x xs)): (n:list A) -> C(n)


null (A:U): list A -> bool
head (A:U): list A -> maybe A
tail (A:U): list A -> maybe (list A)
nth (A:U): nat -> list A -> maybeA
append (A: U): list A -> list A -> list A
reverse (A: U): list A -> list A
map (A B: U): (A -> B) -> list A -> list B
zip (AB: U): list A -> list B -> list (tuple A B)
foldr (AB: U): (A -> B -> B) -> B -> list A -> B
foldl (AB: U): (B -> A -> B) -> B -> list A -> B
switch (A: U): (Unit -> list A) -> bool -> list A
filter (A: U): (A -> bool) -> list A -> list A
length (A: U): list A -> nat
listEq (A: eq): list A.1 -> list A.1 -> bool



data stream (A: U) = cons (x: A) (xs: stream A)







data fin (n: nat)
   = fzero | fsucc (_: fin (pred n))

fz (n: nat): fin (succ n)          = fzero
fs (n: nat): fin n -> fin (succ n) = \(x: fin n) -> fsucc x
```

```
data vector (A: U) (n: nat)
   = nil | cons (_: A) (_: vector A (pred n))


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)
```

```
nat = (X:U) -> (X -> X) -> X -> X
```

$$(X -> X) \text{succ} X \text{zero}$$

```
list (A: U) = (X:U) -> X -> (A -> X) -> X


NAT (A: U) = (X:U) -> isSet X -> X -> (A -> X) -> X


TRUN (A:U) type = (X: U) -> isProp X -> (A -> X) -> X
S1 = (X:U) -> isGroupoid X -> ((x:X) -> Path X x x) -> X
MONOPLE (A:U) = (X:U) -> isSet X -> (A -> X) -> X
NAT = (X:U) -> isSet X -> X -> (A -> X) -> X


upPath     (X Y:U)(f:X->Y)(a:X->X): X -> Y = o X X Y f a
downPath   (X Y:U)(f:X->Y)(b:Y->Y): X -> Y = o X Y Y b f
naturality (X Y:U)(f:X->Y)(a:X->X)(b:Y->Y): U
  = Path (X->Y)(upPath X Y f a)(downPath X Y f b)

unitEnc': U = (X: U) -> isSet X -> X -> X
isUnitEnc (one: unitEnc'): U
  = (X Y:U)(x:isSet X)(y:isSet Y)(f:X->Y) ->
    naturality X Y f (one X x)(one Y y)

unitEnc: U = (x: unitEnc') * isUnitEnc x
unitEncStar: unitEnc = (\(X:U)(_:isSet X) ->
  idfun X,\(X Y: U)(_:isSet X)(_:isSet Y)->refl(X->Y))
unitEncRec  (C: U) (s: isSet C) (c: C): unitEnc -> C
  = \(z: unitEnc) -> z.1 C s c
unitEncBeta (C: U) (s: isSet C) (c: C)
  : Path C (unitEncRec C s c unitEncStar) c = refl C c
unitEncEta (z: unitEnc): Path unitEnc unitEncStar z = undefined
unitEncInd (P: unitEnc -> U) (a: unitEnc): P unitEncStar -> P a
  = subst unitEnc P unitEncStar a (unitEncEta a)
unitEncCondition (n: unitEnc'): isProp (isUnitEnc n)
  =  \(f g: isUnitEnc n) ->
     <h> \(x y: U) -> \(X: isSet x) -> \(Y: isSet y)
  -> \(F: x -> y) -> <i> \(R: x) -> Y (F (n x X R)) (n y Y (F R))
       (<j> f x y X Y F @ j R) (<j> g x y X Y F @ j R) @ h @ i
```

$\mathbb{R}^{n\,11}$

$I = [0, 1]$

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

$i0, i1 : x, y : A$

I

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

$f, g : X \to YH : X \times \ \to Y$

$$\begin{cases} H(x, 0) = f(x), \\ H(x, 1) = g(x). \end{cases}$$

```
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)
```

1213

```
cat: U = (A: U) * (A -> A -> U)
groupoid: U = (X: cat) * isCatGroupoid X
PathCat (X: U): cat = (X,\(x y:X)->Path X x y)
```

$\mathbb{R}$

http://www.cse.chalmers.se/~coquand/Proposal.pdf

```
isCatGroupoid (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)
  * (inv: (x y: C.1) -> C.2 x y -> C.2 y x)
  * (inv_left:  (x y: C.1) (p: C.2 x y) ->
    Path (C.2 x x) (c x y x p (inv x y p)) (id x))
  * (inv_right: (x y: C.1) (p: C.2 x y) ->
    Path (C.2 y y) (c y x y (inv x y p) p) (id 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)))

PathGrpd (X: U)
  : groupoid
  = ((Ob,Hom),id,c,sym X,compPathInv X,compInvPath X,L,R,Q) where
    Ob: U = X
    Hom (A B: Ob): U = Path X A B
    id (A: Ob): Path X A A = refl X A
    c (A B C: Ob) (f: Hom A B) (g: Hom B C): Hom A C
      = comp (<i> Path X A (g@i)) f []
```

∏Σ

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


happly (A B: U) (f g: A -> B) (p: funext_form A B f g) (x: A)
  : Path B (f x) (g x)
  = cong (A -> B) B (\(h: A -> B) -> apply A B h x) f g p


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)
  = \(x:A) -> happly A B f g (funext A B f g p) x


funext_Eta (A B: U) (f g: A -> B) (p: Path (A -> B) f g)
  : Path (Path (A -> B) f g) (funext A B f g (happly A B f g p)) p
  = refl (Path (A -> B) f g) p
```

```
pullback (A B C:U) (f: A -> C) (g: B -> C): U
  = (a: A)
  * (b: B)
  * Path C (f a) (g b)

pb1 (A B C: U) (f: A -> C) (g: B -> C)
  : pullback A B C f g -> A
  = \(x: pullback A B C f g) -> x.1

pb2 (A B C: U) (f: A -> C) (g: B -> C)
  : pullback A B C f g -> B
  = \(x: pullback A B C f g) -> x.2.1

pb3 (A B C: U) (f: A -> C) (g: B -> C)
  : (x: pullback A B C f g) -> Path C (f x.1) (g x.2.1)
  = \(x: pullback A B C f g) -> x.2.2



kernel  (A B: U) (f: A -> B): U
  = pullback A A B f f



hofiber (A B: U) (f: A -> B) (y: B): U
  = pullback A unit B f (\(x: unit) -> y)



pullbackSq (Z A B C: U) (f: A -> C) (g: B -> C) (z1: Z -> A) (z2: Z -> B): U
         = (h: (z:Z) -> Path C ((o Z A C f z1) z) (((o Z B C g z2)) z))
         * isEquiv Z (pullback A B C f g) (induced Z A B C f g z1 z2 h)



completePullback (A B C: U) (f: A -> C) (g: B -> C)
    : pullbackSq (pullback A B C f g) A B C f g (pb1 A B C f g) (pb2 A B C f
     g)




data pushout (A B C: U) (f: C -> A) (g: C -> B)
   = po1 (_: A)
   | po2 (_: B)
   | po3 (c: C) <i> [ (i = 0) -> po1 (f c) ,
                      (i = 1) -> po2 (g c) ]
```

```
isFBundle1 (B: U) (p: B -> U) (F: U): U
  = (_: (b: B) -> isContr (Path U (p b) F))
  * ((x: Sigma B p) -> B)



isFBundle2 (B: U) (p: B -> U) (F: U): U
  = (V: U)
  * (v: surjective V B)
  * ((x: V) -> Path U (p (v.1 x)) F)



im1 (A B: U) (f: A -> B): U = (b: B) * pTrunc ((a:A) * Path B (f a) b)
BAut (F: U): U = im1 unit U (\(x: unit) -> F)
unitIm1 (A B: U) (f: A -> B): im1 A B f -> B = \(x: im1 A B f) -> x.1
unitBAut (F: U): BAut F -> U = unitIm1 unit U (\(x: unit) -> F)

isFBundle3 (E B: U) (p: E -> B) (F: U): U
  = (X: B -> BAut F)
  * (classify B (BAut F) (\(b: B) -> fiber E B p b) (unitBAut F) X) where
  classify (A' A: U) (E': A' -> U) (E: A -> U) (f: A' -> A): U
    = (x: A') -> Path U (E'(x)) (E(f(x)))



isFBundle4 (E B: U) (p: E -> B) (F: U): U
  = (V: U)
  * (v: surjective V B)
  * (v': prod V F -> E)
  * pullbackSq (prod V F) E V B p v.1 v' (\(x: prod V F) -> x.1)




fiber (A B: U) (f: A -> B) (y: B): U = (x: A) * Path B y (f x)
isSingleton (X:U): U = (c:X) * ((x:X) -> Path X c 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



isSurjective (A B: U) (f: A -> B): U
  = (b: B) * pTrunc (fiber A B f b)

surjective (A B: U): U
  = (f: A -> B)
  * isSurjective A B f



isInjective' (A B: U) (f: A -> B): U
  = (b: B) -> isProp (fiber A B f b)

injective (A B: U): U
  = (f: A -> B)
  * isInjective A B f
```

```
isEmbedding (A B: U) (f: A -> B) : U
  = (x y: A) -> isEquiv (Path A x y) (Path B (f x) (f y)) (cong A B f x y)

embedding (A B: U): U
  = (f: A -> B)
  * isEmbedding A B f



isHae (A B: U) (f: A -> B): U
  = (g: B -> A)
  * (eta_: Path (id A) (o A B A g f) (idfun A))
  * (eps_: Path (id B) (o B A B f g) (idfun B))
  * ((x: A) -> Path B (f ((eta_ @ 0) x)) ((eps_ @ 0) (f x)))

hae (A B: U): U
  = (f: A -> B)
  * isHae A B f




iso_Form (A B: U): U = isIso A B -> Path U A B



iso_Intro (A B: U): iso_Form A B



iso_Elim (A B: U): Path U A B -> isIso A B



iso_Comp (A B : U) (p : Path U A B)
  : Path (Path U A B) (iso_Intro A B (iso_Elim A B p)) p



iso_Uniq (A B : U) (p: isIso A B)
  : Path (isIso A B) (iso_Elim A B (iso_Intro A B p)) p




univ_Formation (A B: U): U = equiv A B -> Path U A B



equivToPath (A B: U): univ_Formation A B
  = \(p: equiv A B) -> <i> Glue B [(i=0) -> (A,p),
    (i=1) -> (B, subst U (equiv B) B B (<_>B) (idEquiv B)) ]
```

```
pathToEquiv (A B: U) (p: Path U A B) : equiv A B
  = subst U (equiv A) A B p (idEquiv A)
```

```
eqToEq (A B : U) (p : Path U A B)
  : Path (Path U A B) (equivToPath A B (pathToEquiv A B p)) p
  = <j i> let Ai: U = p@i in Glue B
    [ (i=0) -> (A,pathToEquiv A B p),
      (i=1) -> (B,pathToEquiv B B (<k> B)),
      (j=1) -> (p@i,pathToEquiv Ai B (<k> p @ (i \/ k))) ]
```

```
transPathFun (A B : U) (w: equiv A B)
  : Path (A -> B) w.1 (pathToEquiv A B (equivToPath A B w)).1
```

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

$$\mathrm{i0}, \mathrm{i1} : x, y : A$$

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

```
data S2
   = point
   | surf <i j> [ (i = 0) -> point, (i = 1) -> point,
                  (j = 0) -> point, (j = 1) -> point ]
                  (j = 0) -> point, (j = 1) -> point ]
```

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

```
data pTrunc (A: U) -- (-1)-trunc, mere proposition truncation
  = pinc (a: A)
  | pline (x y: pTrunc A) <i>
        [ (i = 0) -> x,
          (i = 1) -> y ]

data sTrunc (A: U) -- (0)-trunc, set truncation
  = sinc (a: A)
  | sline (a b: sTrunc A)
          (p q: Path (sTrunc A) a b) &lt;i j&gt;
        [ (i = 0) -> p @ j,
          (i = 1) -> q @ j,
          (j = 0) -> a,
          (j = 1) -> b ]

data gTrunc (A: U) -- (1)-trunc, groupoid truncation
  = ginc   (a: A)
  | gline  (a b: gTrunc A)
           (p q: Path (gTrunc A) a b)
           (r s: Path (Path (gTrunc A) a b) p q) &lt;i j k&gt;
         [ (i = 0) -> r @ j @ k,
           (i = 1) -> s @ j @ k,
           (j = 0) -> p @ k,
           (j = 1) -> q @ k,
           (k = 0) -> a,
           (k = 1) -> b ]
```

```
data quot (A: U) (R: A -> A -> U)
  = inj (a: A)
  | quoteq (a b: A) (r: R a b) &lt;i&gt;
         [ (i = 0) -> inj a,
           (i = 1) -> inj b ]

data setquot (A: U) (R: A -> A -> U)
  = quotient (a: A)
  | identification (a b: A) (r: R a b) &lt;i&gt;
                 [ (i = 0) -> quotient a,
                   (i = 1) -> quotient b ]
```

```
  | setTruncation  (a b: setquot A R)
                   (p q: Path (setquot A R) a b) &lt;i j&gt;
                 [ (i = 0) -> p @ j,
                   (i = 1) -> q @ j,
                   (j = 0) -> a,
                   (j = 1) -> b ]
```

```
storage: U -> U = list
```

Σ

```
process : U
  = (protocol state: U)
  * (current: prod protocol state)
  * (act: id (prod protocol state))
  * (storage (prod protocol state))
```

```
spawn (protocol state: U) (init: prod protocol state)
      (action: id (prod protocol state)) : process
    = (protocol,state,init,action,nil)
```

```
protocol  (p: process): U = p.1
state     (p: process): U = p.2.1
signature (p: process): U = prod p.1 p.2.1
current   (p: process):          signature p  = p.2.2.1
action    (p: process):      id (signature p) = p.2.2.2.1
trace     (p: process): storage (signature p) = p.2.2.2.2
```

$$P \times S \to SP \times S \to P \times S$$

```
receive (p: process) : protocol p = axiom
```

```
send (p: process) (message: protocol p) : unit = axiom
```

```
execute (p: process) (message: protocol p) : process
  = let step: signature p = (action p) (message, (current p).2)
    in (protocol p, state p, step, action p, cons step (trace p))
```

14
15
16

https://n2o.dev/articles/pi.htm
https://n2o.space/articles/bpe.htm
https://sample.n2o.dev

$$\Sigma_{A:\mathcal{U}} A \to A \to \mathcal{U}\mathcal{U}\mathrm{pr}_1\mathrm{Ob}\mathrm{pr}_2\mathrm{Hom}(a,b)a,b : \mathrm{Ob}$$

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

$$\mathrm{CHom}_C(a,b)a,b : \mathrm{Ob}_C\mathrm{idHom}_C(x,x)$$

$$\mathrm{COb}_C a,b : \mathrm{Ob}_C\mathrm{Hom}_C(a,b)a : \mathrm{Ob}_C 1_a : \mathrm{Hom}_C(a,a)a,b,c : \mathrm{Ob}_C\mathrm{Hom}_C(b,c) \to \mathrm{Hom}_C(a,b) \to \mathrm{Hom}_C(a,c)g \circ fa,b : \mathrm{Ob}_C f : \mathrm{Hom}_C(a,b)f = 1_b \circ ff = f \circ 1_a a,b,c,d : Af : \mathrm{Hom}_C(a,b)g : \mathrm{Hom}_C(b,c)h : \mathrm{Hom}_C(c,d)h \circ (g \circ f) = (h \circ g) \circ f$$

$$a,b : \mathrm{Ob}\mathrm{Hom}_C(a,b)$$

```
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)))
```

ObHom

```
carrier (C: precategory): U = C.1.1
hom     (C: precategory) (a b: carrier C): U = C.1.2 a b
path    (C: precategory) (x: carrier C): hom C x x = C.2.1 x
compose (C: precategory) (x y z: carrier C)
        (f: hom C x y) (g: hom C y z): hom C x z = C.2.2.1 x y z f g
```

$\mathrm{Ob}_C \Pi_{x,y:\mathrm{Ob}_C} \mathrm{isContr}(\mathrm{Hom}_C(x,y))$

$\mathrm{Ob}_C \Pi_{x,y:\mathrm{Ob}_C} \mathrm{isContr}(\mathrm{Hom}_C(y,x))$

```
isInitial (C: precategory) (x: carrier C): U
  = (y: carrier C) -> isContr (hom C x y)
isTerminal (C: precategory) (y: carrier C): U
  = (x: carrier C) -> isContr (hom C x y)
initial (C: precategory): U
  = (x: carrier C) * isInitial  C x
terminal(C: precategory): U
  = (y: carrier C) * isTerminal C y
```

$ABF : A \to BF_{Ob} : \mathrm{Ob}_h A \to \mathrm{Ob}_B a, b : \mathrm{Ob}_A F_{Hom} : \mathrm{Hom}_A(a,b) \to \mathrm{Hom}_B(F_{Ob}(a), F_{Ob}(b))a : \mathrm{Ob}_A F_{Ob}(1_a) = 1_{F_{Ob}}(a)a, b, c : \mathrm{Ob}_A f : \mathrm{Hom}_A(a,b)g : \mathrm{Hom}_A(b,c)F(g \circ f) = F_{Hom}(g) \circ F_{Hom}(f)$

```
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)))
```

$F, G : C \to D\gamma : F \to Gx : C\gamma_a : \mathrm{Hom}_D(F(x), G(x))x, y : Cf : \mathrm{Hom}_C(x,y)$ $G(f) \circ \gamma_x = \gamma_y \circ F(g)$

```
isNaturalTrans (C D: precategory)
               (F G: catfunctor C D)
               (eta: (x: carrier C) -> hom D (F.1 x) (G.1 x)): U
  = (x y: carrier C) (h: hom C x y) ->
     Path (hom D (F.1 x) (G.1 y))
     (compose D (F.1 x) (F.1 y) (G.1 y) (F.2.1 x y h) (eta y))
     (compose D (F.1 x) (G.1 x) (G.1 y) (eta x) (G.2.1 x y h))

ntrans (C D: precategory) (F G: catfunctor C D): U
  = (eta: (x: carrier C) -> hom D (F.1 x) (G.1 x))
  * (isNaturalTrans C D F G eta)
```

```
extension (C C' D: precategory)
    (K: catfunctor C C') (G: catfunctor C D) : U
  = (F: catfunctor C' D)
  * (ntrans C D (compFunctor C C' D K F) G)
```

$f : Hom_A(a,b) g : Hom_A(b,a) 1_a =_\eta g \circ f f \circ g =_\epsilon 1_b = g a, b : A a = b \to iso_A(a,b)$

```
iso (C: precategory) (A B: carrier C): U
  = (f: hom C A B)
  * (g: hom C B A)
  * (eta: Path (hom C A A) (compose C A B A f g) (path C A))
  * (Path (hom C B B) (compose C B A B g f) (path C B))
```

$a : Ob_C \Pi_{A:Ob_C} isContr \Sigma_{B:Ob_C} iso_C(A,B)$

```
isCategory (C: precategory): U
  = (A: carrier C) -> isContr ((B: carrier C) * iso C A B)
    category: U = (C: precategory) * isCategory C
```

```
Product    (X Y: precategory) : precategory
Coproduct  (X Y: precategory) : precategory
```

CC^{op}

```
opCat (P: precategory): precategory
```

```
sliceCat (C D: precategory)
    (a: carrier (opCat C))
    (F: catfunctor D (opCat C))
  : precategory
  = cosliceCat (opCat C) D a F
```

```
cosliceCat (C D: precategory)
  (a: carrier C)
  (F: catfunctor D C) : precategory
```

```
initArr (C D: precategory)
  (a: carrier C)
  (F: catfunctor D C): U = initial (cosliceCat C D a F)

termArr (C D: precategory)
  (a: carrier (opCat C))
  (F: catfunctor D (opCat C)): U = terminal (sliceCat C D a F)
```

$$\mathrm{Ob} = \top \mathrm{Hom} = \top$$

```
unitCat: precategory
```

```
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))
```

```
Functions (X Y: U) (Z: isSet Y): precategory
   = ((Ob,Hom),id,c,HomSet,L,R,Q) where
  Ob: U = X -> Y
  Hom (A B: Ob): U = id (X -> Y)
  id (A: Ob): Hom A A = idfun (X -> Y)
  c (A B C: Ob) (f: Hom A B) (g: Hom B C): Hom A C = idfun (X -> Y)
```

```
  HomSet (A B: Ob): isSet (Hom A B) = setFun Ob Ob (setFun X Y Z)
  L (A B: Ob) (f: Hom A B): Path (Hom A B) (c A A B (id A) f) f = axiom
  R (A B: Ob) (f: Hom A B): Path (Hom A B) (c A B B f (id B)) f = axiom
  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)) = axiom
```

```
Cat: precategory = ((Ob,Hom),id,c,HomSet,L,R,Q) where
  Ob: U = precategory
  Hom (A B: Ob): U = catfunctor A B
  id (A: Ob): catfunctor A A = idFunctor A
  c (A B C: Ob) (f: Hom A B) (g: Hom B C): Hom A C
   = compFunctor A B C f g
  HomSet (A B: Ob): isSet (Hom A B) = axiom
  L (A B: Ob) (f: Hom A B): Path (Hom A B) (c A A B (id A) f) f = axiom
  R (A B: Ob) (f: Hom A B): Path (Hom A B) (c A B B f (id B)) f = axiom
  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)) = axiom
```

```
Func (X Y: precategory): precategory
   = ((Ob,Hom),id,c,HomSet,L,R,Q) where
  Ob: U = catfunctor X Y
  Hom (A B: Ob): U = ntrans X Y A B
  id (A: Ob): ntrans X Y A A = axiom
  c (A B C: Ob) (f: Hom A B) (g: Hom B C): Hom A C = axiom
  HomSet (A B: Ob): isSet (Hom A B) = axiom
  L (A B: Ob) (f: Hom A B): Path (Hom A B) (c A A B (id A) f) f = axiom
  R (A B: Ob) (f: Hom A B): Path (Hom A B) (c A B B f (id B)) f = axiom
  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)) = axiom
```

$k(k-1)01$

```
equiv: U
functor (C D: cat): U
ntrans (C D: cat) (F G: functor C D): U
modification (C D: cat) (F G: functor C D) (I J: ntrans C D F G): U
```

```
Cat2 : U
  = (Ob: U)
  * (Hom:  (A B: Ob) -> U)
  * (Hom2: (A B: Ob) -> (C F: Hom A B) -> U)
  * (id:     (A: Ob) -> Hom A A)
  * (id2:    (A: Ob) -> (B: Hom A A) -> Hom2 A A B B)
  * (c:  (A B C: Ob) (f: Hom A B) (g: Hom B C) -> Hom A C)
  * (c2:   (A B: Ob) (X Y Z: Hom A B)
    (f: Hom2 A B X Y) (g: Hom2 A B Y Z) -> Hom2 A B X Z)
```

$A \xrightarrow{f} C \xleftarrow{g} BA \times_C Bpb_1 : \times_C \to Apb_2 : \times_C \to B$

$$A \times_C B \xrightarrow{pb_2} B \downarrow{\scriptstyle pb_1} \quad A \xrightarrow{f} C$$

$(\times_C, pb_1, pb_2)(D, q_1, q_2)u : D \to \times_C pb_1 \circ u = q_1 pb_2 \circ q_2$

```
homTo  (C: precategory) (X: carrier C): U
  = (Y: carrier C) * hom C Y X
cospan (C: precategory): U
  = (X: carrier C) * (_: homTo C X) * homTo C X
cospanCone (C: precategory) (D: cospan C): U
  = (W: carrier C) * hasCospanCone C D W
cospanConeHom (C: precategory) (D: cospan C)
    (E1 E2: cospanCone C D) : U
  = (h: hom C E1.1 E2.1) * isCospanConeHom C D E1 E2 h
isPullback (C: precategory) (D: cospan C) (E: cospanCone C D) : U
  = (h: cospanCone C D) -> isContr (cospanConeHom C D h E)
hasPullback (C: precategory) (D: cospan C) : U
  = (E: cospanCone C D) * isPullback C D E
```

$ABF : A \to BF_{Ob} : Ob_h A \to Ob_B a, b : Ob_A F_{Hom} : Hom_A(a, b) \to Hom_B(F_{Ob}(a), F_{Ob}(b))a : Ob_A F_{Ob}(1_a) = 1_{F_{Ob}}(a)a, b, c : Ob_A f : Hom_A(a, b)g : Hom_A(b, c)F(g \circ f) = F_{Hom}(g) \circ F_{Hom}(f)$

```
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)))
```

Ob_C

$$\prod_{x,y:\mathrm{Ob}_C} \mathrm{isContr}(\mathrm{Hom}_C(y,x)).$$

```
isTerminal (C: precategory) (y: carrier C): U
  = (x: carrier C) -> isContr (hom C x y)
terminal (C: precategory): U
  = (y: carrier C) * isTerminal C y
```

∞

$\mathrm{PROP} x, y : P x = y$

$$\mathrm{isProp}(P) = \prod_{x,y:P} (x = y).$$

$x, y : A p, q : x =_A y p = q$

$x, y : A p, q : x =_A y r, s : p =_{=_A} q r = s$

$\mathrm{SETSET} x, y : A p, q : x = y p = q$

$$\mathrm{isSet}(A) = \prod_{x,y:A} \prod_{p,q:x=y} (p = q).$$

```
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 }

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

Π

```
setPi (A: U) (B: A -> U) (h: (x: A) -> isSet (B x)) (f g: Pi A B)
      (p q: Path (Pi A B) f g)
   : Path (Path (Pi A B) f g) p q
```

$\Sigma\Sigma$

```
setSig (A:U) (B: A -> U) (base: isSet A)
       (fiber: (x:A) -> isSet (B x)) : isSet (Sigma A B)
```

11

```
data unit = tt
unitRec (C: U) (x: C): unit -> C = split tt -> x
unitInd (C: unit -> U) (x: C tt): (z:unit) -> C z
  = split tt -> x
```

SetHomΠ

```
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))
```

$AS \in ASSSSSS$

```
Structure topology (A : Type) := {
          open :> (A -> Prop) -> Prop;
          empty_open: open (empty _);
          full_open:  open (full _);
          inter_open: forall u,
                      open u -> forall v, open v
                             -> open (inter A u v) ;
          union_open: forall s, (subset _ s open)
                             -> open (union A s) }.
```

1

$$R \subset \mathrm{Hom}_C(,U), U \in C,$$

$R \subset \mathrm{Hom}_C(,U)\phi : V \to UC$

$$\phi^{-1}(R) = \{\gamma : W \to V \| \phi \cdot \gamma \in R\}$$

$VR, R' \subset \mathrm{Hom}_C(,U)\phi^{-1}(R')\phi : V \to URR'\mathrm{Hom}_C(,U)U \in C$

$\mathrm{Ob}_C\{f_i : U_i \to U\}_{i\in I} g : V \to U\{h : V_j \to V\}_{j\in J} h_j \circ gf_i \overset{V_j U_i}{\underset{V_j U}{\rightrightarrows}} V_i$

```
Co (C: precategory) (cod: carrier C) : U
  = (dom: carrier C)
  * (hom C dom cod)

Delta (C: precategory) (d: carrier C) : U
  = (index: U)
  * (index -> Co C d)

Coverage (C: precategory): U
  = (cod: carrier C)
  * (fam: Delta C cod)
  * (coverings: carrier C -> Delta C cod -> U)
  * (coverings cod fam)
```

CCC

$$\{\phi_\alpha : U_\alpha \to U\}, U \in C,$$

$\phi_\alpha : U_\alpha \to U\psi : V \to UCV \times_U U_\alpha \to VV\{\phi_\alpha : U_\alpha \to U\}\{\gamma_{\alpha,\beta} : W_{\alpha,\beta} \to U_\alpha\}\alpha$

$$W_{\alpha,\beta} \xrightarrow{\gamma_{\alpha,\beta}} U_\alpha \xrightarrow{\phi_\alpha} U$$

$\{1 : U \to U\}U \in C$

```
site (C: precategory): U
  = (C: precategory) * Coverage C
```

$CC^{op} \to \mathrm{Set}$

```
presheaf (C: precategory): U
  = catfunctor (opCat C) Set
```

C

$F : C^{op} \to \mathbf{Set}$

$$F(U) \to \underset{V\to U\in R}{\overleftarrow{\lim}} F(V)$$

$R \subset \mathrm{Hom}_C(,U)$

$$\mathrm{Hom}_C(\mathrm{Hom}_C(,U), F) \to \mathrm{Hom}_C(R, F)$$

```
sheaf (C: precategory): U
  = (S: site C)
  * presheaf S.1
```

$(C, J)CJ$

$CR \to ER \to E \to QC$

CD

$$f : (C) \to (D)$$

$f_* : (C) \to (D) f^* : (D) \to (C) f^* f_* f^* f^* f_* f^*$

$ES(p^*, p_*) : E \to Sp^! \vdash p_* p^! \dashv p_* p^* p^! p_!$

$$\int \dashv \flat \dashv \sharp$$

$f : Y \to ZXg_1, g_2 : X \to Y$

$$f \circ g_1 = f \circ g_2 \to g_1 = g_2.$$

$XHom(X,)Hom(X, Y) \to Hom(X, Z)$

```
mono (P: precategory) (Y Z: carrier P) (f: hom P Y Z): U
   = (X: carrier P) (g1 g2: hom P X Y)
  -> Path (hom P X Z) (compose P X Y Z g1 f)
                      (compose P X Y Z g2 f)
  -> Path (hom P X Y) g1 g2
```

$Ctrue : 1 \to \Omega 1U \to X\chi_U : X \to \Omega_{\chi_U^*(\Omega)}^{k}$

```
subobjectClassifier (C: precategory): U
  = (omega: carrier C)
  * (end: terminal C)
  * (trueHom: hom C end.1 omega)
  * (chi: (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,chi 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) (chi V X j) k)
```

CMLTT

```
isCCC (C: precategory): U
  = (Exp:   (A B: carrier C) -> carrier C)
  * (Prod:  (A B: carrier C) -> carrier C)
  * (Apply: (A B: carrier C) -> hom C (Prod (Exp A B) A) B)
  * (P1:    (A B: carrier C) -> hom C (Prod A B) A)
  * (P2:    (A B: carrier C) -> hom C (Prod A B) B)
  * (Term:  terminal C)
  * unit
```

ΠΣSETisSetpropPi**MLTT**

```
cartesianClosure : isCCC Set
  = (expo,prod,appli,proj1,proj2,term,tt) where
    exp (A B: SET): SET = (A.1   -> B.1, setFun A.1 B.1 B.2)
    pro (A B: SET): SET = (prod A.1 B.1, setSig A.1 (\(_ : A.1)
                          -> B.1) A.2 (\(_ : A.1) -> B.2))
    expo:  (A B: SET) -> SET = \(A B: SET) -> exp A B
    prod:  (A B: SET) -> SET = \(A B: SET) -> pro A B
    appli: (A B: SET) -> hom Set (pro (exp A B) A) B
        = \(A B: SET) -> \(x:(pro(exp A B)A).1)-> x.1 x.2
    proj1: (A B: SET) -> hom Set (pro A B) A
        = \(A B: SET) (x: (pro A B).1) -> x.1
    proj2: (A B: SET) -> hom Set (pro A B) B
        = \(A B: SET) (x: (pro A B).1) -> x.2
    unitContr (x: SET) (f: x.1 -> unit) : isContr (x.1 -> unit)
      = (f, \(z: x.1 -> unit) -> propPi x.1 (\(_:x.1)->unit)
           (\(x:x.1) -> propUnit) f z)
    term: terminal Set = ((unit,setUnit),
           \(x: SET) -> unitContr x (\(z: x.1) -> tt))
```

```
Topos (cat: precategory) : U
  = (cartesianClosure: isCCC cat)
  * subobjectClassifier cat
```

```
internal : Topos Set
         = (cartesianClosure,hasSubobject)
```

$$Cb : Ob_C C \downarrow bf : a \to bf^* : C \downarrow b \to c \downarrow a \sum_f \prod_f$$

$S^{n-1} \hookrightarrow D^n D^n S^{n-1}$

$Xf : S^{n-1} \to X$

$$\begin{array}{ccc} S^{n-1} & \xrightarrow{k} & X \\ D^n & \xrightarrow{g} & \bigcup_f D^n \end{array}$$

$X \cup_f D^n f$

$-1 \varnothing \leqslant nXXn-1$

$Xcolimit(X_i)X_{-1} = \varnothing \hookrightarrow X_0 \hookrightarrow X_1 \hookrightarrow X_2 \hookrightarrow ...XX_i \leqslant nX_{i+1}X_i$

$$\varnothing \hookrightarrow X_0 \hookrightarrow X_1 \hookrightarrow X_2 \hookrightarrow ...X$$

$X_i \leqslant i$

$(A, a)A : Ua : A$

```
pointed: U = (A: U) * A
point (A: pointed): A.1 = A.2
space (A: pointed): U = A.1
```

$$\Omega(A, a) =_{def} ((a =_A a), refl_A(a)).$$

```
omega1 (A: pointed) : pointed
  = (Path (space A) (point A) (point A), refl A.1 (point A))
```

$$\begin{cases} \Omega^0(A, a) =_{def} (A, a) \\ \Omega^{n+1}(A, a) =_{def} \Omega^n(\Omega(A, a)) \end{cases}$$

```
omega : nat -> pointed -> pointed = split
  zero -> idfun pointed
  succ n -> \(A: pointed) -> omega n (omega1 A)
```

$$\pi_n S^m = \|\Omega^n(S^m)\|_0.$$

```
piS (n: nat): (m: nat) -> U = split
   zero   -> sTrunc (space (omega n (bool,false)))
   succ x -> sTrunc (space (omega n (Sn (succ x),north)))
```

$$\Omega(S^1) = \mathbb{Z}$$

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

loopS1 : U = Path S1 base base

encode (x:S1) (p:Path S1 base x)
  : helix x
  = subst S1 helix base x p zeroZ

decode : (x:S1) -> helix x -> Path S1 base x = split
  base -> loopIt
  loop @ i -> rem @ i where
    p : Path U (Z -> loopS1) (Z -> loopS1)
      = <j> helix (loop1@j) -> Path S1 base (loop1@j)
    rem : PathP p loopIt loopIt
      = corFib1 S1 helix (\(x:S1)->Path S1 base x) base
        loopIt loopIt loop1 (\(n:Z) ->
        comp (<i> Path loopS1 (oneTurn (loopIt n))
             (loopIt (testIsoPath Z Z sucZ predZ
                      sucpredZ predsucZ n @ i)))
             (<i>(lem1It n)@-i) [])

loopS1eqZ : Path U Z loopS1
  = isoPath Z loopS1 (decode base) (encode base)
    sectionZ retractZ
```

$S^3 S^1 S^2 S^3 \mathbb{R}^3 S^0 S^1 S^3 S^7$
$S^3 S^2$

S^3

$S^3 \mathbb{R}^4$

$$S^3 = \{(x_0, x_1, x_2, x_3) \in \mathbb{R}^4 : \sum_{i=0}^{3} x_i^2 = 1\};$$

$\mathbb{H}$

$$S^3 = \{x \in \mathbb{H} : \|x\| = 1\}.$$

$S^3(\eta, \theta_1, \theta_2)$

$$\begin{cases} x_0 = \cos(\theta_1)\sin(\eta), \\ x_1 = \sin(\theta_1)\sin(\eta), \\ x_2 = \cos(\theta_2)\cos(\eta), \\ x_3 = \sin(\theta_2)\cos(\eta). \end{cases}$$

$\eta \in [0, \frac{\pi}{2}] \theta_{1,2} \in [0, 2\pi]$

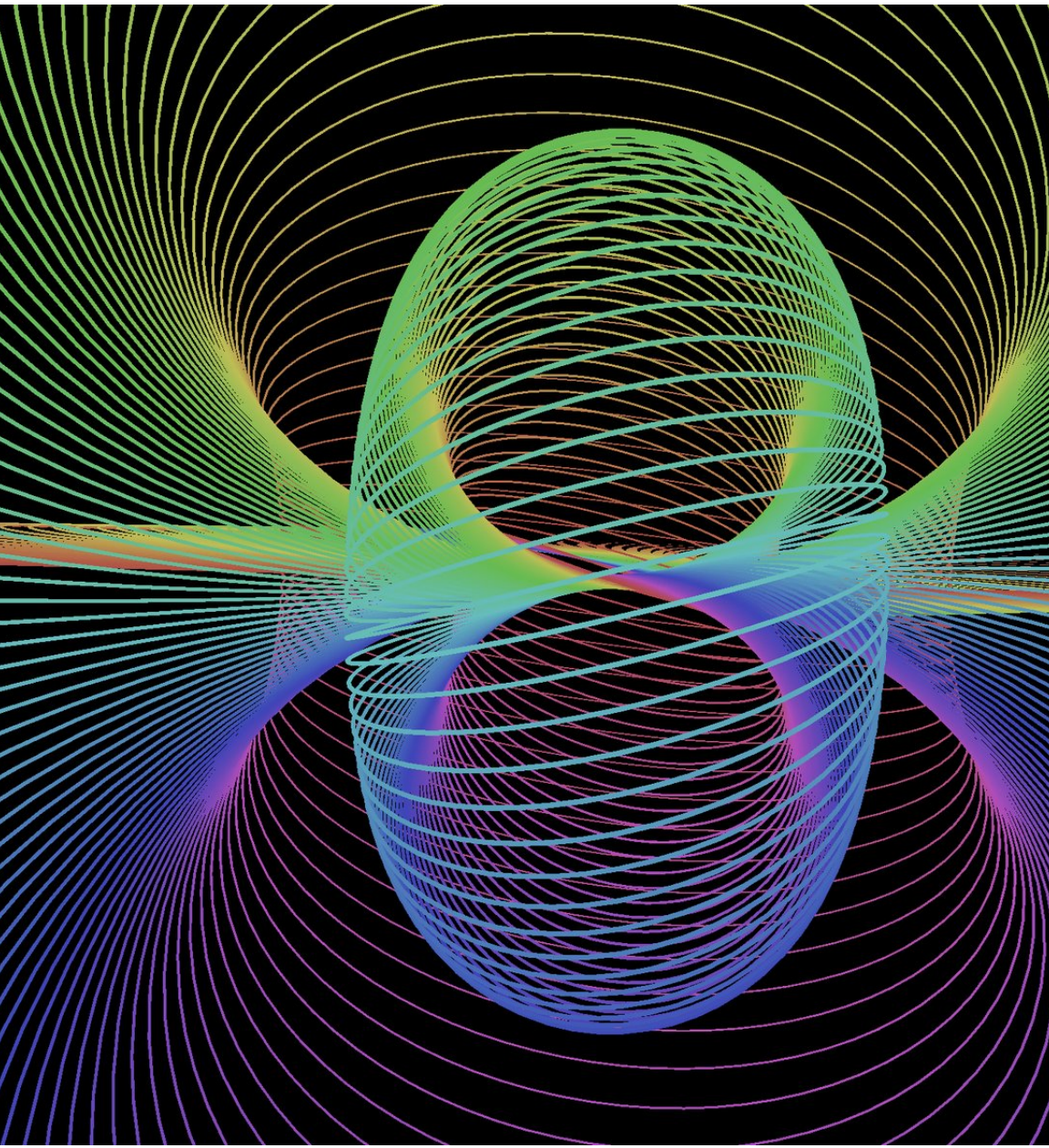
$S^2 S^2 \theta_2$

$$\begin{cases} x = \sin(2\eta)\cos(\theta_1), \\ y = \sin(2\eta)\sin(\theta_1), \\ z = \cos(2\eta). \end{cases}$$

```
var fiber = new THREE.Curve(),
    color = sphericalCoords.color;

fiber.getPoint = function(t) {
    var eta = sphericalCoords.eta,
        phi = sphericalCoords.phi,
        theta = 2 * Math.PI * t;
    var x1 = Math.cos(phi+theta) * Math.sin(eta/2),
        x2 = Math.sin(phi+theta) * Math.sin(eta/2),
        x3 = Math.cos(phi-theta) * Math.cos(eta/2),
        x4 = Math.sin(phi-theta) * Math.cos(eta/2);
    var m = mag([x1,x2,x3]),
        r = Math.sqrt((1-x4)/(1+x4));
        return new THREE.Vector3(r*x1/m,r*x2/m, r*x3/m);
    };
```

S^3

```
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

total : U = (c : S2) * H c
```



A

$$H_A = \begin{cases} A : U \\ e : A \\ \mu : A \to A \to A \\ \beta : (a : A) \to \Sigma(\mu(e, a) = a)(\mu(a, e) = a) \end{cases}$$

$(S^0, S^1, p, S^1)(S^1, S^3, p, S^2)(S^3, S^7, p, S^4)(S^7, S^{15}, p, S^8)$

$\phi : S^{2n-1} \to S^n \phi \mathrm{cofib}(\phi) = S^n \bigcup_\phi \mathbb{D}^{2n}$

$$H^k(\mathrm{cofib}(\phi), \mathbb{Z}) = \begin{cases} \mathbb{Z} \mathrm{fork} = n, 2n \\ 0 \mathrm{otherwise} \end{cases}$$

$\alpha, \beta n 2 n h(\phi) \alpha \beta \alpha \sqcup \alpha = h(\phi) \cdot \beta h(\phi) \phi$

1

OHTS

```
module buddhism where
import path
```

```
concept (o: U): U
  = o -> U
```

```
nondual (o: U) (p: concept o): U
  = (x y: o) -> Path U (p x) (p y)
```

```
allpaths (o: U): U
  = (x y: o) -> Path o x y
```

=U=O

```
encode (o:U): ((p: concept o) -> nondual o p) -> allpaths o
  = \(nd: (p: concept o) -> nondual o p) (a b: o)
  -> coerce(Path o a a)(Path o a b)(nd(\(z:o)->Path o a z)a b)(refl o a)
```

```
decode (o:U): allpaths o -> ((p: concept o) -> nondual o p)
  = \(all: allpaths o)(p: concept o)(x y: o) -> cong o U p x y (all x y)
```


https://favonia.org/files/thesis.pdf

```
CoInductive Co (E : Effect.t) : Type -> Type :=
    | Bind : forall (A B : Type), Co E A -> (A -> Co E B) -> Co E B
    | Split : forall (A : Type), Co E A -> Co E A -> Co E A
    | Join : forall (A B : Type), Co E A -> Co E B -> Co E (A * B).
    | Ret : forall (A : Type) (x : A), Co E A
    | Call : forall (command : Effect.command E),
                   Co E (Effect.answer E command)

Definition run (argv : list LString.t): Co effect unit :=
    ido! log (LString.s "What is your name?") in
    ilet! name := read_line in
    match name with
      | None => ret tt
      | Some name => log (LString.s "Hello " ++ name ++ LString.s "!")
    end.

Parameter infinity : nat.
Definition eval {A} (x : Co effect A) : Lwt.t A := eval_aux infinity x.

Fixpoint eval_aux {A} (s: nat) (x: Co effect A) : Lwt.t A :=
  match s with
    | O => error tt
      | S s =>
    match x with
      | Bind _ _ x f => Lwt.bind   (eval_aux s x) (fun v_x => eval_aux s (f
     v_x))
      | Split _ x y  => Lwt.choose (eval_aux s x) (eval_aux s y)
      | Join _ _ x y => Lwt.join   (eval_aux s x) (eval_aux s y)
      | Ret _ v => Lwt.ret v
      | Call c => eval_command c
    end
  end.

CoFixpoint handle_commands : Co effect unit :=
    ilet! name := read_line in
    match name with
      | None => ret tt
      | Some command =>
        ilet! result := log (LString.s "Input: "
                     ++ command ++ LString.s ".")
     in handle_commands
    end.

Definition launch (m: list LString.t -> Co effect unit): unit :=
    let argv := List.map String.to_lstring Sys.argv in
    Lwt.launch (eval (m argv)).

Definition corun (argv: list LString.t): Co effect unit :=
    handle_commands.

Definition main := launch corun.
```

```
String: Type = List Nat
data IO: Type =
     (getLine: (String -> IO) -> IO)
     (putLine: String -> IO)
     (pure: () -> IO)
```

```
-- IOI/@: (r: U) [x: U] [[s: U] -> s -> [s -> #IOI/F r s] -> x] x
   \ (r : *)
-> \/ (x : *)
-> (\/ (s : *)
   -> s
   -> (s -> #IOI/F r s)
   -> x)
-> x

-- IOI/F
   \ (a : *)
-> \ (State : *)
-> \/ (IOF : *)
-> \/ (PutLine_ : #IOI/data -> State -> IOF)
-> \/ (GetLine_ : (#IOI/data -> State) -> IOF)
-> \/ (Pure_ : a -> IOF)
-> IOF

-- IOI/MkIO
   \ (r : *)
-> \ (s : *)
-> \ (seed : s)
-> \ (step : s -> #IOI/F r s)
-> \ (x : *)
-> \ (k : forall (s : *) -> s -> (s -> #IOI/F r s) -> x)
-> k s seed step
```

```
-- Morte/corecursive
( \ (r: *1)
 -> ( (((#IOI/MkIO r) (#Maybe/@ #IOI/data)) (#Maybe/Nothing #IOI/data))
    ( \ (m: (#Maybe/@ #IOI/data))
     -> (((((#Maybe/maybe #IOI/data) m) ((#IOI/F r) (#Maybe/@ #IOI/data)))
           ( \ (str: #IOI/data)
            -> ((((#IOI/putLine r) (#Maybe/@ #IOI/data)) str)
                (#Maybe/Nothing #IOI/data))))
        (((#IOI/getLine r) (#Maybe/@ #IOI/data))
         (#Maybe/Just #IOI/data))))))
```

```
copure() ->
    fun (_) -> fun (IO) -> IO end end.

cogetLine() ->
    fun(IO) -> fun(_) ->
        L = ch:list(io:get_line("> ")),
        ch:ap(IO,[L]) end end.

coputLine() ->
    fun (S) -> fun(IO) ->
        X = ch:unlist(S),
        io:put_chars(": "++X),
        case X of "0\n" -> list([]);
                      _ -> corec() end end end.

corec() ->
    ap('Morte':corecursive(),
        [copure(),cogetLine(),coputLine(),copure(),list([])]).


> om_extract:extract("priv/normal/IOI").
ok
> Active: module loaded: {reloaded,'IOI'}

> om:corec().
> 1
: 1
> 0
: 0
#Fun<List.3.113171260>
```

```
-- IO/@
   \ (a : *)
-> \/ (IO : *)
-> \/ (GetLine_ : (#IO/data -> IO) -> IO)
-> \/ (PutLine_ : #IO/data -> IO -> IO)
-> \/ (Pure_ : a -> IO)
-> IO

-- IO/replicateM
   \ (n: #Nat/@)
-> \ (io: #IO/@ #Unit/@)
-> #Nat/fold n (#IO/@ #Unit/@)
               (#IO/[>>] io)
               (#IO/pure #Unit/@ #Unit/Make)


-- Morte/recursive
((#IO/replicateM #Nat/Five)
 ((((#IO/[>>=] #IO/data) #Unit/@) #IO/getLine) #IO/putLine))
```

```
pure() ->
    fun(IO) -> IO end.

getLine() ->
    fun(IO) -> fun(_) ->
        L = ch:list(io:get_line("> ")),
        ch:ap(IO,[L]) end end.

putLine() ->
    fun (S) -> fun(IO) ->
        io:put_chars(": "++ch:unlist(S)),
        ch:ap(IO,[S]) end end.

rec() ->
    ap('Morte':recursive(),
        [getLine(),putLine(),pure(),list([])]).



> om:rec().
> 1
: 1
> 2
: 2
> 3
: 3
> 4
: 4
> 5
: 5
#Fun<List.28.113171260>
```