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

$\star : \star\star ::: \star_{\omega},), , .$

$$\begin{tikzcd}
 & \lambda\omega \arrow[rr] & & \lambda P\omega \arrow[dd] \\
\lambda 2 \arrow[ur] \arrow[rr] \arrow[dd] & & \lambda P 2 \arrow[ur] \arrow[dd] & \\
 & \lambda\underline{\omega} \arrow[rr, dotted] & & \lambda P\underline{\omega} \\
\lambda_{\to} \arrow[ur] \arrow[rr] & & \lambda P \arrow[ur] &
\end{tikzcd}$$

$[0, 1]$

910

1112

1314

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

1516

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

34

567

12

34

567

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

89

1011

$\pi\pi$

$O_{CPS}\pi$

O_{CPS}[12][13]

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/tphols09-lisp.pdf

$O_\lambda \lambda O_\pi \pi O_\mu O_\Pi O_\Sigma O_= O_{W} O_I$

1

$\Pi\Sigma{=}{+}\bot\top\mathsf{N}\mathsf{U}_i\mathsf{El}$

https://github.com/guillaumebrunerie/initiality

$\Pi x : op(o) : Up : concept(o)$

$Ob : UHom : Ob \to Ob \to U$

$id \circ$

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

$p : E \to Bf : Y \to Xp(f)Bp(f) = \mathrm{id}_B$

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

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

$B^{\to}Ba : \mathrm{Ob}_{B}^{\to} = \mathrm{Hom}_B(x, y)\mathrm{Hom}_{B}^{\to} = [f : \mathrm{Hom}_B, g : \mathrm{Hom}_B]B$

$$\begin{tikzcd}
x \arrow[r, "f"] \arrow[d, "a"'] & f(x) \arrow[d, "f(a)"] \\
y \arrow[r, "f"'] & f(y)
\end{tikzcd}$$

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

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

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

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

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

$\mathrm{Psh}(B)\mathrm{Fib}(B)$

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

$\Pi\Sigma$

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

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

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

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

```
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)
```

$C - C -$

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

$$\begin{tikzcd}
y(D) \arrow[r, "y"] & Q \arrow[d, "\alpha"] \\
y(C) \arrow[u, "yp_x"] \arrow[r, "x"] & P
\end{tikzcd}$$

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

$$\begin{tikzcd}
F_! Tm(C) \arrow[r, "\phi_{Tm}"] & Tm(D) \\
F_! Ty(C) \arrow[u, "F_! p(C)"] \arrow[r, "\phi_{Ty}"] & Ty(D) \arrow[u, "p(D)"]
\end{tikzcd}$$

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

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

$\beta\eta$

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

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

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

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

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

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

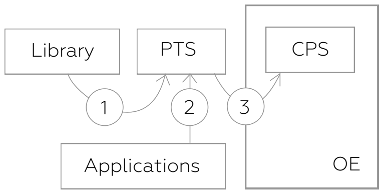
$O_{MLTT-80}$

$\beta\eta\beta\eta$

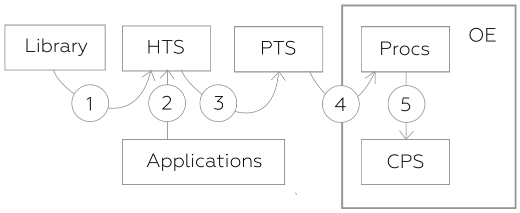
$$O_{\infty} : O_{CPS} \to O_{PTS} \to O_{MLTT-80} \to O_{HTS} \to \ldots$$

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

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



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



O_{CPS}

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

$O_\lambda O_\pi O_\mu$

O_{CPS}

```
data CPS = lambda (_: church CPS)
         | process (_: milner CPS)
         | tensor (_: futhark CPS)
```

$\lambda O_{CPS} maybe$

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

O_λ

```
data church = var (x: nat)
            | lam (l: nat) (d: cps)
            | app (f a: cps)
```

$O_\pi O_*$

```
data milner (lang: U)
   = 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

```
data futhark (lang: U)
   = iota (cursor: lang)
   | map (cursor: lang)
   | fold (size: nat)
   | scan (cursor: lang)
   | for (cursor: lang)
   | while (cursor: lang)
   | concat (cursor: lang)
   | zip (cursor: lang)
   | transpose (cursor: lang)
```

Π

$\quad\Sigma$

$\quad\mathsf{N} \to \mathsf{U}$

O_{PTS}

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

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

```
data PTS = forall (_: Forall PTS)
```

O_{Π}

```
data Forall (lang: U)
   = 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)
```

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

$O_{MLTT-75}$

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

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

```
data MLTT
   = forall (_: Forall MLTT)
   | sigma (_: Sigma MLTT)
   | id (_: Id MLTT)
```

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

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

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

```
data Id (lang: U)
   = 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

$O_{MLTT-80}$

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

$O_{=}O_{\Sigma}O_{\Pi}O_{0}O_{1}O_{2}O_{W}$

$O_{MLTT-80}$

```
data MLTT-80
   = forall (_: Forall MLTT-80)
   | sigma (_: Sigma MLTT-80)
   | id (_: Id MLTT-80)
   | 0 (_: Empty MLTT-80)
   | 1 (_: Unit MLTT-80)
   | 2 (_: Bool MLTT-80)
   | W (_: W MLTT-80)

data W (lang: U) =
   | W_Form (n: name) (a b: lang)
   | W_Sup (a b: lang)
   | W_Ind (a b c: lang)

data 2 (lang: U) =
   | 2_bool | 2_true | 2_false
   | 2_Ind (a: lang)

data 1 (lang: U) =
   | 1_unit | 1_star
   | 1_Ind (a: lang)

data 0 (lang: U) =
   | 0_Ind (a: lang)
```

O_{PM}

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

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

O_{PM}

```
data PM = forall (_: Forall PM)
   | sigma (_: Sigma PM)
   | id (_: Id PM)
   | inductive (_: InductiveSchemes PM)
```

```
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)
   | com (n: name) (t: tele A) (dim: list name)
         (s: list (prod (prod name bool) A))
```

$O_{*}O_{*}$

```
data InductiveSchemes (lang: U)
   = data (n: name) (t: tele lang) (labels: list (label lang))
   | case (n: name) (t: lang) (branches: list (branch lang))
   | constructor (n: name) (args: list lang)
```

O_{HTS}

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

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

```
data HTS = forall (_: Forall HTS)
         | sigma (_: Sigma HTS)
         | id (_: Id HTS)
         | homotopy (_: Homotopy HTS)
```

$O_{MLTT-80}$

O_I

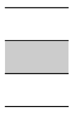
```
data Homotopy (lang: U)
   = pretype (n: nat)
   | path (A x y: lang)
   | path_lambda (name: name) (a: lang)
   | path_app (f a: lang)
   | interval | zero | one
   | meet (a b: lang) | join (a b: lang) | neg (e: lang)
   | transp (a b c: lang) | hcomp (a b: lang)
   | glue (a b c: lang) | Glue (a b: lang) | unglue (a b: lang)
```

$O_{HTS} O_= O_I$

O<span style="font-variant:small-caps">cps</span>

```
data cps = var (x: nat)
         | lam (l: nat) (d: cps)
         | app (f a: cps)
```

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
```

OCPS

```
data Lazy = Defer (otree: NodeId) (a: AST) (cont: Cont)
          | Continuation (otree: NodeId) (a: AST) (cont: Cont)
          | Return (a: AST)
          | Start
```

```
data Cont = 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)
```

OCPS

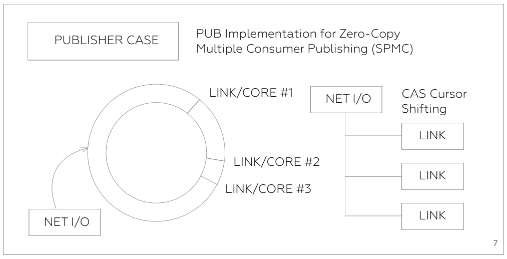
```
E: V | A | C
NC: ";" = [] | ";" m:NL = m
FC: ";" = [] | ";" m:FL = m
EC: ";" = [] | ";" m:EL = m
NL: NAME | o:NAME m:NC = Cons o m
FL: E | o:E | m:FC = Cons o m
EL: E | EC  | o:E m:EC = Cons o m
C: N | c:N a:C = Call c a
N: NAME | S | HEX | L | F
L: "(" ")" = [] | "([" c:NL "]" m:FL ")" = Table c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

```
data AST      = Atom (a: Scalar)
              | Vector (a: Vec AST)

data Value    = Nil
              | SymbolInt (a: u16)
              | SequenceInt (a: u16)
              | Number (a: i64)
              | Float (a: f64)
              | VecNumber (Vec i64)
              | VecFloat (Vec f64)

data Scalar   = 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)
```

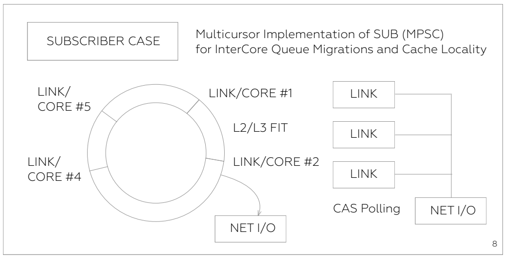
1

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

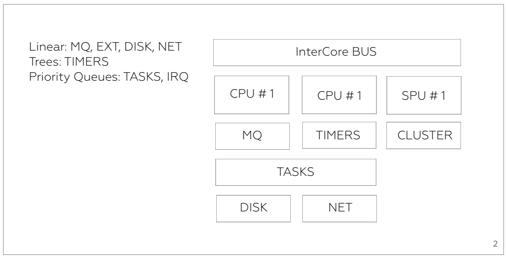
https://github.com/voxoz/kernel



PUB Implementation for Zero-Copy Multiple Consumer Publishing (SPMC)



Multicursor Implementation of SUB (MPSC) for InterCore Queue Migrations and Cache Locality



Linear: MQ, EXT, DISK, NET
Trees: TIMERS
Priority Queues: TASKS, IRQ

```
pub [ capacity ]
```

```
sub [ publisher ]
```

```
spawn [ core ; program ; cursors ]
```

```
snd [ writer ; data ]
```

```
rcv [ reader ]
```

O_{CPS}

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

```
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,
}
```

```rust
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,
}
```

$e_i : O_{n+1} \to O_n O_{CPS} = O_0$

O<span style="font-variant:small-caps">pts</span>[2]O<span style="font-variant:small-caps">cchm</span>[3]

[2] https://github.com/groupoid/om
[3] https://github.com/groupoid/cubical

1

2

3

4

1

2

3

4

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

5

6

[7]F_{ω}

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'); }});};
```

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

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

[4]

ω

morte[5]cubical[6]caramel[7]

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

$\Pi\lambda\beta\eta$

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

πλμ

∞

8

ω

ω

ω

```
I := #list #nat
U := * + * . #nat
O := U + I + ( O ) + O O + O ⟶ O
   + λ ( I : O ) ⟶ O
   + ∀ ( I : O ) ⟶ O
```

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

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

∞_{ω}[9]Fixpointremote

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

$U_0 U_1 U_2 U_3$

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

$U : U$

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

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

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

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

`list Sigma`

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

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

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

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

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

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

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

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

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

```
PTS (A: U): U
  = (Form: (A -> U) -> U)
  * (Ctor: (B: A -> U) -> ((a: A) -> B a) -> (Pi A B))
  * (Elim: (B: A -> U)(a: A) -> (Pi A B) -> B a)
  * (Beta: (B: A -> U)(a: A)(f: Pi A B)-> Equ (B a)(Elim B a (Ctor B f))(f a))
  * (Eta: (B: A -> U)(a: A)(f: Pi A B)-> Equ (Pi A B) f (\(x:A) -> f x)) * 1
```

$O_{=}O_I$

```
remoteremote

type (:star,N)     D ⟶ (:star,N+1)
     (:var,N,I)    D ⟶ :true = proplists:is_defined N B, om:keyget N D I
     (:remote,N)   D ⟶ om:cache (type N D)
     (:pi,N,0,I,O) D ⟶ (:star,h(star(type I D)),star(type O [(N,norm I)|D]))
     (:fn,N,0,I,O) D ⟶ let star (type I D), NI = norm I
                         in (:pi,N,0,NI,type(O,[(N,NI)|D]))
     (:app,F,A)    D ⟶ let T = type(F,D),
                             (:pi,N,0,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)
     (:remote,X)    N P ⟶ (:remote,X)
     (:pi,N,0,I,O)  N P ⟶ (:pi,N,0,sh I N P,sh O N P+1)
     (:fn,N,0,I,O)  N P ⟶ (:fn,N,0,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
     (:remote,X)    N V L ⟶ (:remote,X)
     (:pi,N,0,I,O)  N V L ⟶ (:pi,N,0,sub I N V L,sub O N (sh V N 0) L+1)
     (:pi,F,X,I,O)  N V L ⟶ (:pi,F,X,sub I N V L,sub O N (sh V F 0) L)
     (:fn,N,0,I,O)  N V L ⟶ (:fn,N,0,sub I N V L,sub O N (sh V N 0) L+1)
     (:fn,F,X,I,O)  N V L ⟶ (:fn,F,X,sub I N V L,sub O N (sh V F 0) 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)
     (:remote,N)    → cache (norm N [])
     (:pi,N,0,I,O)  → (:pi,N,0,norm I,norm O)
     (:fn,N,0,I,O)  → (:fn,N,0,norm I,norm O)
     (:app,F,A)     → case norm F of
                         (:fn,N,0,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
     (:remote,N)      (:remote,N)      → true
     (:pi,N1,0,I1,O1) (:pi,N2,0,I2,O2) →
          let :true = eq I1 I2
           in eq O1 (subst (shift O2 N1 0) N2 (:var,N1,0) 0)
     (:fn,N1,0,I1,O1) (:fn,N2,0,I2,O2) →
          let :true = eq I1 I2
           in eq O1 (subst (shift O2 N1 0) N2 (:var,N1,0) 0)
     (: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
```

PTS$_\infty$λλ

```
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)]}]))
                     _ ⟶ []
```

∞

O_{PTS}

```
def := data id tele = sum + id tele : exp = exp +
       id tele : exp where def
exp := cotele*exp + cotele → exp + exp → exp + (exp) + app + id +
       (exp,exp) + \ cotele → exp + split cobrs + exp .1 + exp .2

  0 := #empty         imp    := [ import id ]
brs := 0 + cobrs      tele   := 0 + cotele
app := exp exp        cotele := ( exp : exp ) tele
 id := [ #nat ]       sum    := 0 + id tele + id tele | sum
ids := [ id ]         br     := ids → exp
cod := def dec        mod    := module id where imp def
dec := 0 + codec      cobrs  := | br brs
```

```
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)
                         | com (n: name) (t: tele A) (dim: list name)
                               (s: list (prod (prod name bool) A))
```

```
data ind (lang: U)
   = datum  (n: name) (t: tele lang) (labels:   list (label lang))
   | case   (n: name) (t: lang)      (branches: list (branch lang))
   | ctor   (n: name)                (args:     list lang)
```

$$F_A(X) = 1 + A \times X F_A(X) = A + X \times X 1 A F_A(X) = A \times X$$
$$\mu X \to 1 + X$$
$$\mu X \to 1 + A \times X$$
$$\mu X \to 1 + X \times X + X$$
$$\nu X \to A \times X$$
$$\nu X \to 1 + A \times X$$
$$\mu X \to \mu Y \to 1 + X \times Y = \mu X = \text{List} X$$

$\prod\prod$`Fixpoint` `funExthomotopy`

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

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

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

O_{PTS}

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

```
data list (A: U) = cons (x: A) (cs: list A) | nil
```

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

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

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

W012

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

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

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

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

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

```
  sys := [ sides ]          side := (id=0)⟶exp+(id=1)⟶exp
 form := form\/f1+f1+f2    sides := #empty+cos+side
  cos := side,side+side,cos  mod := module id where imps dec
   f1 := f1/\f2               f2 := -f2+id+0+1
  imp := import id                 brs := #empty+cobrs
  app := exp exp             tel := #empty+cotel
 imps := #list imp         cotel := (exp:exp) tel
   id := #list #nat          dec := #empty+codec
   u2 := glue+unglue+Glue                    u1 := fill+comp
  ids := #list id             br := ids⟶exp+ids@ids⟶exp
codec := def dec
cobrs := | br brs
  sum := #empty+id tel+id tel|sum+id tel<ids>sys
  def := data id tel=sum+id tel:exp=exp+id tel:exp where def
  exp := cotel*exp+cotel⟶exp+exp⟶exp+(exp)+id
         (exp,exp)+\cotele⟶exp+split cobrs+exp.1+exp.2+
         ⟨ids⟩exp+exp@form+app+u2 exp exp sys+u1 exp sys
```

|⟨⟩\⟶moduleimportdatasplitwherecompfillGlueglueunglue
.1.2,

cubical

```
data hts (lang: U)
   = pre (n: nat)
   | path (A x y: lang)
   | plam (name: name) (a: lang)
   | papp (f a: lang)
   | interval
   | zero
   | one
   | meet (a b: lang)
   | join (a b: lang)
   | neg (e: lang)
   | comp (a b: lang)
   | fill (a b c: lang)
   | glue (a b c: lang)
   | glue-1 (a b: lang)
   | unglue-1 (a b: lang)
```

$\mathbf{Henk} U_i \Pi \tau = \mathbf{typePerAnders} \Sigma \tau \Pi$

ω

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

$\alpha\beta$

$= V_i U_i \Pi$

τ

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

Σ

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

$e_i : O_{n+1} \rightarrow O_n O_{CPS} = O_0$
O_{PTS}[10]O_{CCHM}[11]

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

ΠΣ=

123

$^{4}\infty$

PerAndersAndersAnders

	$\bot$	
	$\top$	
	$A \vee B$	
	$A \wedge B$	
	$A \Rightarrow B$	
$x)$	$\exists_{x:A} B(x)$	
$x)$	$\forall_{x:A} B(x)$	
	$=_A$	A^I

Anders

PerAnders

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)
```

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

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

```
lambda (A B: U) (b: B): A -> B = \(x: A) -> b
lam (A: U) (B: A -> U) (b: (a: A) -> B a): Pi A B = \(x: A) -> b x
```

$\prod$

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

```
apply (A B: U) (f: A -> B) (a: A) : B = f a
app (A: U) (B: A -> U) (a: A) (f: Pi A B): B a = f a
```

Π

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

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

Π

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

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

$\Pi\Sigma$

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

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

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

```
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)
```

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

$\Pi\Pi$

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

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

$$\begin{tikzcd} p^{-1}(U_b) \arrow[r, "f"] \arrow[d, "p"] & U_b \times F \arrow[dl, "pr_1"] \\ U_b & \end{tikzcd}$$

$F app : F \times B \to E$

$$F \times B \xrightarrow{app} E \xrightarrow{pr_1} B$$

$pr_1 pr_1 app Set_{/B} F A A \times B \to E Set_{/B} A \to F$

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

$f : (x : A) \to B(x) ap_f : x =_A y \to f(x) =_{B(x)} f(y) fcong$

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

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

$f, g : (x : A) \to B(x)$

```
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
```

Σ

$\Sigma\Sigma$

Σ

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

Σ

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

Σ

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

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

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
```

Σ

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

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

Σ

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

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

$x : A y : B R(x,y) f : A \rightarrow B x : A R(x, f(x))$

```
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)
```

$[0,1]$[5][6][7][8][9]

```
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`$\lambda(i : I) \rightarrow a$

```
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
```

$$\lambda(i : I) \rightarrow \overset{\text{comp}}{[illegible]}$$

---

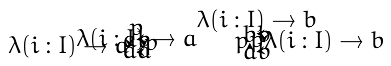
https://5ht.co/cubicaltt.pdf
https://5ht.co/cctt.pdf
http://www.cse.chalmers.se/~coquand/mod1.pdf

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

```
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
```

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



```
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)
```

$[0,1]\lambda\to$

```
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)
```

$U_{n \in N} U_0$

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

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

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

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

$U_i, i \in N \in$

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

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

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

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

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

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

$\Pi\Sigma$

$\Pi\Sigma\Sigma$

```
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-comp₂ : 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)
```

$(\mu L_A, in) L_A(X) = 1 + (AX) \mu L_A = List(A) nil : 1 \longrightarrow List(A) cons : A \times List(A) \longrightarrow List(A) nil = in \circ inl cons = in \circ inr in = [nil, cons]$

```
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
```

$(X- > X) succ X zero$

```
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 }
```

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

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

```
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)
```

[12][13]

```
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 []
```

ΠΣ

```
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]
```

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

```
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
```

$$P \times S \longrightarrow SP \times S \longrightarrow P \times S$$

```
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

$\Sigma_{A:U} A \to A \to U U pr_1 Ob pr_2 Hom(a,b) a,b : Ob$

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

$C Hom_C(a,b) a,b : Ob_C id Hom_C(x,x)$

$C Ob_C a,b : Ob_C Hom_C(a,b) a : Ob_C 1_a : Hom_C(a,a) a,b,c : Ob_C Hom_C(b,c) \to Hom_C(a,b) \to Hom_C(a,c) g \circ f a,b : Ob_C f : Hom_C(a,b) f = 1_b \circ f f = f \circ 1_a a,b,c,d : A f : Hom_C(a,b) g : Hom_C(b,c) h : Hom_C(c,d) h \circ (g \circ f) = (h \circ g) \circ f$

$a,b : Ob Hom_C(a,b)$

```
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)
```

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

```
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
```

$C C^{op}$

```
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)
```

$A \xrightarrow{f} C \xleftarrow{g} B A \times_C B pb_1 : \times_C \to A pb_2 : \times_C \to B$

$$\begin{tikzcd}
A \times_C B \arrow[r, "pb_2"] \arrow[d, "pb_1"] & B \arrow[d, "g"] \\
A \arrow[r, "f"] & C
\end{tikzcd}$$

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

```
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
```

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

```
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
```

∞

PROP$x, y : Px = y$

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

$x, y : Ap, q : x =_A yp = q$

$x, y : Ap, q : x =_A yr, s : p =_{=_A} qr = s$

SETSET$x, y : Ap, q : x = yp = q$

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

```
  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

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

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

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

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

$\mathrm{Ob}_C \{f_i : U_i \to U\}_{i \in I} g : V \to U \{h : V_j \to V\}_{j \in J} h_j \circ g f_i$
$$\begin{tikzcd} V_j \arrow[r, "k"] \arrow[d, "h"'] & U_i \arrow[d, "f_i"] \\ V \arrow[r, "g"'] & U \end{tikzcd}$$

```
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)
```

$C C C$

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

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

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

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

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

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

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

C

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

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

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

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

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

$(C, J) C J$

$C R \to E R \to E \to Q C$

$C D$

$$f : \mathbf{Sh}(C) \to \mathbf{Sh}(D)$$

$f_* : \mathbf{Sh}(C) \to \mathbf{Sh}(D) f^* : \mathbf{Sh}(D) \to \mathbf{Sh}(C) f^* f_* f^* f^* f_* f^*$

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

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

$f : Y \to Z X g_1, g_2 : X \to Y$

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

$X \mathrm{Hom}(X,) \mathrm{Hom}(X, Y) \to \mathrm{Hom}(X, Z)$

```
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
```

$C \mathrm{true} : 1 \to \Omega 1 U \to X \chi_U : X \to \Omega$. $U \xrightarrow{k} 1$ true $X \Omega \chi_U \Omega$

```
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)
```

$C \mathrm{MLTT}$

```
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
```

$\Pi\Sigma$SETisSetpropPi**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)
```

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

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

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

$$\begin{tikzcd}S^{n-1} \arrow[r, hook, "k"] \arrow[d, hook] & X \arrow[d, hook] \\ D^n \arrow[r, hook, "g"] & X \cup_f D^n\end{tikzcd}$$

$X \cup_f D^n f$

$-1 \varnothing \leqslant nXXn-1$
$X \mathrm{colimit}(X_i) X_{-1} = \varnothing \hookrightarrow X_0 \hookrightarrow X_1 \hookrightarrow X_2 \hookrightarrow \dots XX_i \leqslant nX_{i+1}X_i$

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

$X_i \leqslant i$

$(A, a) A : U a : A$

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

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

```
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
```

$S^3 S^1 S^2 S^3 R^3 S^0 S^1 S^3 S^7$
$S^3 S^2$

S^3

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

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

$\mathbb{H}$

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

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

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

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

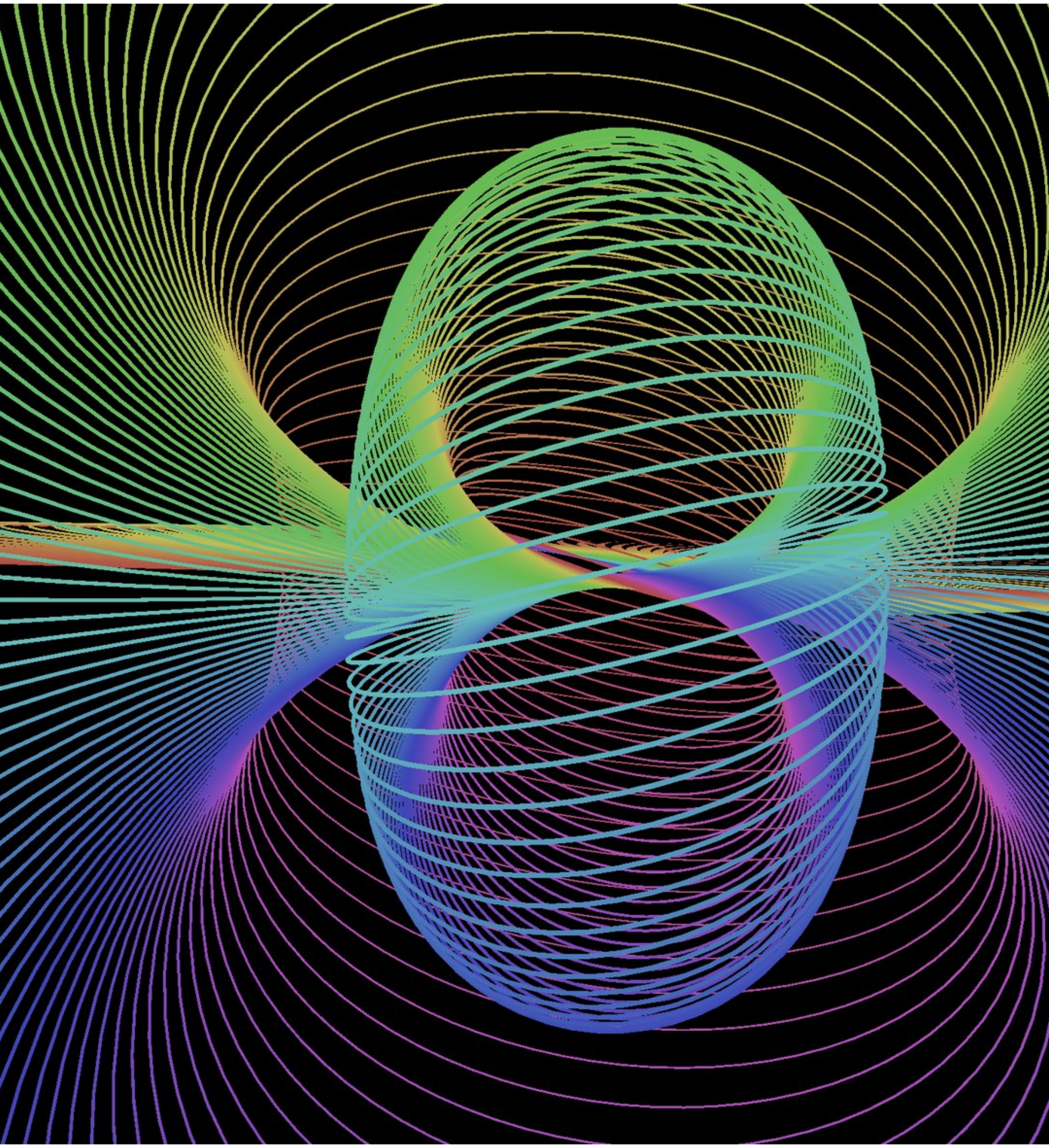
$S^2 S^2 \theta_2$

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

```
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

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

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

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

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

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

1

O_{HTS}

```
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
```

$U(py)x =_o y.(coerce)(cong).$

```
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)
```

1

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>
```