MLTT — CT — Proof Theory

$x : A$ — x is a object of type A
$x = y : A$ — x and y are definitionaly equal objects of type A

$Nat : U\ n$ — constant functor
$List\ (A : U\ n) : U\ n$ — functor
$List\ Nat : U\ n$ — constant functor

$$List\ A = 1 \longrightarrow List\ A \longrightarrow List\ A \longleftarrow A : U$$
$$nil : 1 \longrightarrow List\ A$$
$$cons : A \longrightarrow List\ A \longrightarrow List\ A$$

$$Nat = 1 \longrightarrow N \longrightarrow N : U$$
$$zero : 1 \longrightarrow N$$
$$succ : N \longrightarrow N$$

Pure Type System

Infinity Topoi

Grothendieck Universum

$S\ (n : nat) = U\ n$

$A_1\ (n\ m : nat) = U\ n : U\ m$ where $m > n$ — cumulative

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

$R_1\ (m\ n : nat) = U\ m \longrightarrow U\ n : U\ (\max\ m\ n)$ — predicative

U_0 — propositions

$A_2\ (n : nat) = U\ n : U\ (n + 1)$ — non-cumulative

U_1 — sets

$R_2\ (m\ n : nat) = U\ m \longrightarrow U\ n : U\ n$ — impredicative

U_2 — types

U_3 — sorts

$Prop = Large\ \Omega_0 = U_0$

$\boldsymbol{\Sigma} = Large\ \Omega_2 = U_2$

Pi Type

Functional Completness

```
data O₁ := U : nat → O₁
         | Var: Ident → O₁
         | App: O₁ → O₁ → O₁
         | Lambda: Binder → O₁ → O₁ → O₁
         | Arrow: O₁ → O₁ → O₁
         | Pi: name → O₁ → O₁ → O₁.
```

∀ x: A, B x : U — formation rule
λ x: A, b : B x — introduction
app f a : B x — elimination
app (**λ** o:A, b) a = b [a/o] : B x — computation

```
record Pi (A: Type) :=
    intro: (A → Type) → Type :=
    fun: (B: A → Type) → ∀ (a: A) → B a → intro B
    app: (B: A → Type) → intro B → ∀ (a: A) → B a
    app-fun (B: A → Type) (f: ∀ (a: A) → B a): ∀ (a: A) → app (fun f) a ==> f a
    fun-app (B: A → Type) (p: intro B): fun (λ (a: A) → app p a) ==> p
```

Sigma Types

Contextual Completness

```
data O₂ := O₁
          | Sigma: name → O₂ → O₂ → O₂
          | Pair: O₂ → O₂ → O₂
          | Fst: O₂ → O₂
          | Snd: O₂ → O₂.
```

$\Sigma\, x : A, B\, x : U$ — formation rule

$\mathrm{pair}\ (x : A)\ (y : B\, x)$ — introduction

$\mathrm{pr}_1\ s : A$ — elimination

$\mathrm{pr}_2\ s : B\, x$ — elimination

Identity Type a la Martin-Löf

```
record Id (A: Type): Type :=
    Id: A → A → Type
    refl (a: A): Id a a
    Predicate: ∀ (x,y: A) → Id x y → Type
    Forall (C: Predicate): ∀ (x,y: A) → ∀ (p: Id x y) → C x y p
    Δ (C: Predicate): ∀ (x: A) → C x x (refl x)
    axiom-J (C: Predicate): Δ C → Forall C
    computation (C: Predicate) (t: Δ C): ∀ (x: A) → (J C t x x (refl x)) ==> (t x) )

record Subst (A: Type): Type :=
    intro (P (a: A): Type) (a1,a2: A) : Id a1 a2 → P a1 → P a2 :=
            Id.axiom-J (λ a1 a2 p12 → P a1 → P a2))
```

K UIP Congruence

```
record UIP (A: Type): Type :=
    intro (A: Type) (a,b: A) (x,y: Id a b) : Id (Id A a b) x y)

record K (A: Type): Type :=
    PredicateK: ∀ (a: A) → Id a a → Type
    ForallK (C:PredicateK): ∀ (a: A) → ∀ (p: Id a a) → C a p
    ΔK (C: PredicateK) : ∀ (a: A) → C a (Id.refl a)
    axiom-K (C: Predicate): ΔK C → ForallK C)

 define Respect (A,B: Type) (C: A → Type) (D: B → Type) (R: A → B → Prop)
    (Ro: ∀ (x: A) (y: B) → C x → D y → Prop) : (∀ (x: A) → C x) → (∀ (x: B) → D x) → Prop
  := λ (f,g: Type → Type) → (∀ (x,y: Type) → R x y) → Ro x y (f x) (g y)
```

Category

```
record Cat: U :=
    Ob: U
    Hom: (dom,cod: Ob) → Setoid
    Id:     (x: Ob) → Hom x x
    Comp:  (x,y,z: Ob) → Hom x y → Hom y z → Hom x z
    Dom1ₒ:  (x,y: Ob) (f: Hom x y) → (Hom.Equ x y (Comp x x y id f) f)
    Cod1ₒ:  (x,y: Ob) (f: Hom x y) → (Hom.Equ x y (Comp x y y f id) f)
    Subst1: (x,y,z: Ob) → Proper (Respect Equ (Respect Equ Equ)) (Comp x y z)
    Substₒ: (x,y,z: Ob) (f1,f2: Hom x y) (g1,g2: Hom y z)
          → (Hom.Equ x y f1 f2) → (Hom.Equ y z g1 g2)
          → (Hom.Equ x z (Comp x y z f1 g1) (Comp x y z f2 g2))
    Assocₒ: (x,y,z,w: Ob) (f: Hom x y) (g: Hom y z) (h: Hom z w)
          → (Hom.Equ x w (Comp x y w f (Comp y z w g h))
                 (Comp x z w (Comp x y z f g) h))
```

Setoid

```
record Setoid: Type :=
        Carrier: Type
        Equ: Carrier → Carrier → Prop
        Refl: (x: Carrier) → Equ x x
        Trans: (x₁,x₂,x₃: Carrier) → Equ x₁ x₂ → Equ x₂ x₃ → Equ x₁ x₃
        Sym: (x₁,x₂: Carrier) → Equ x₁ x₂ → Equ x₂ x₁
```

Application 1. Logic

```
data Proper (A: Type) (R: A → A → Prop) (m: A): Prop := intro: R m m
data Inhabited (A: Type): Prop := intro: A → Inhabited A
data True: Prop := intro: () → True
data False: Prop := ()
data Eq (A : Type): A → A → Type := refl: ∀ (x: A) → Eq A x x
data Exists (A: Type): A → Type → Type := intro: ∀ (P: A → Type) →
                                                    ∀ (x: A) → P x → Exists A P

record SKI := S : (p → q → r) → (p → q) → p → r
              K :  p → (q → p)
              I : p -> p
```

Application 2. Control

```
record pure (P: Type → Type) (A: Type): Type := return: P A
record functor (F: Type → Type) (A,B: Type): Type := map: (A → B) → F A → F B
record applicative (F: Type → Type) (A,B: Type): Type :=
          pure: pure F A
          functor: functor F A B
          ap: F (A → B) → F A → F B

record monad (F: Type → Type) (A,B: Type): Type :=
          pure: pure F A
          functor: functor F A B
          join: F (F A) → F B
```

Core

```
import Strings.String.
data  Loc: Type := intro: string → nat → nat → Loc.

define Ident          := string.
define Label          := string.
define Binder         := prod Ident Loc.
define Branch         := prod Label (prod (list Binder) Type).
define NamedRec  := list (prod Binder Type).
define NamedSum := list (prod Binder NamedRec).
define DecList        := list (prod Binder (prod Type Type)).

data Decls            := Intro:  DecList → Decls
                       | Opaque:   Binder → Decls
                       | Transparent: Binder → Decls.
```

Infinity Language

```
data Infinity := O₂
    | Where:   MLTT → Decls → MLTT
    | Con:     Label → list MLTT → MLTT
    | Split:   Loc → list Branch → MLTT
    | Sum:     Binder → NamedSum → MLTT
    | HIT:     HomotopyCalculus → MLTT
    | PI:      PiCalculus → MLTT
    | EFF:     EffectCalculus → MLTT
    | STREAM:  StreamCalculus → MLTT.
```

Homotopy Calculus

```
import core.

data HomotopyCalculus :=
 | Id            | Refl         | Inh          | Inc
 | Squash        | InhRec       | TransU       | TransInvU
 | TransURef     | Singl        | MapOnPath    | AppOnPath
 | HExt          | EquivEq      | EquivEqRef   | TransUEquivEq
 | IdP           | MapOnPathD   | IdS          | MapOnPathS
 | AppOnPathD    | Circle       | Base         | HLoop
 | CircleRec     | I            | I0           | I1
 | Line          | IntRec       | Undef: Loc → HomotopyCalculus.
```

Equiv Squash Id Inh Inc

```
Id : (A : U) (a b : A) → U
IdP : (A B : U) → Id U A B → A → B → U
refl : (A : U) (a : A) → Id A a a
inh : U → U
inc : (A : U) → A → inh A
squash : (A : U) → prop (inh A)
inhrec : (A : U) (B : U) (p : prop B) (f : A → B) (a : inh A) → B
contrSingl : (A : U) (a b : A) (p : Id A a b) → Id (singl A a) (a, refl A a) (b, p)
equivEq : (A B : U) (f : A → B) (s : (y : B) → fiber A B f y)
        (t : (y : B) → (v : fiber A B f y) →
        Id (fiber A B f y) (s y) v) → Id U A B
equivEqRef : (A : U) → (s : (y : A) → pathTo A y) →
           (t : (y : A) → (v : pathTo A y) →
           Id (pathTo A y) (s y) v) →
           Id (Id U A A) (refl U A) (equivEq A A (id A) s t)
```

Proposition Fibration
Path Singleton

```
id : (A : U) → A → A
id A a = a
sId : (A : U) (a : A) → pathTo A a
sId A a = (a, refl A a)
singl : (A : U) → A → U
singl A a = Sigma A (Id A a)
pathTo : (A:U) → A → U
pathTo A = fiber A A (id A)

prop : U → U
prop A = (a b : A) → Id A a b
Sigma : (A : U) (B : A → U) → U
Sigma A B = (x : A) * B x
fiber : (A B : U) (f : A → B) (y : B) → U
fiber A B f y = Sigma A ( λ x → Id B (f x) y)

IdS : (A : U) (F : A → U) (a0 a1 : A) (p : Id A a0 a1) → F a0 → F a1 → U
IdS A F a0 a1 p = IdP (F a0) (F a1) (mapOnPath A U F a0 a1 p)
```

Transport

```
transport : (A B : U) → Id U A B → A → B
transpInv : (A B : U) → Id U A B → B → A
transportRef : (A : U) (a : A) → Id A a (transport A A (refl U A) a)
transpEquivEq : (A B : U) → (f : A → B) (s : (y : B) → fiber A B f y) →
        (t : (y : B) → (v : fiber A B f y) → Id (fiber A B f y) (s y) v) →
        (a : A) → Id B (f a) (transport A B (equivEq A B f s t) a)
```

App

```
appOnPath : (A B : U) (f g : A → B) (a b : A) (q : Id (A → B) f g) (p : Id A a b) → Id B (f a) (g b)
appOnPathD :  (A : U) (F : A → U) (f g : (x : A) → F x) → Id ((x : A) → F x) f g →
        (a0 a1 : A) (p : Id A a0 a1) → IdS A F a0 a1 p  (f a0) (g a1)
```

Map

```
mapOnPath : (A B : U) (f : A → B) (a b : A) (p : Id A a b) → Id B (f a) (f b)
mapOnPathD : (A:U) (F: A → U) (f: (x:A) -> F x) (a0 a1: A) (p: Id A a0 a1) → IdS A F a0 a1 p  (f a0) (f a1)
mapOnPathS : (A:U) (F: A → U) (C: U) (f: (x:A) → F x → C)
        (a0 a1 : A) (p : Id A a0 a1) (b0 : F a0) (b1 : F a1)
        (q : IdS A F a0 a1 p b0 b1) → Id C (f a0 b0) (f a1 b1)
```

Extensionality

```
funHExt : (A : U) (B : A → U) (f g : (a : A) → B a) →
          ((x y : A) → (p : Id A x y) → IdS A B x y p (f x) (g y)) → Id ((y : A) → B y) f g
```

Interval

```
record I : U :=
    I0 : I
    I1 : I
    line : Id I I0 I1
    rec : (F : I → U) (s : F I0) (e : F I1) (l : IdS I F I0 I1 line s e) (x : I) → F x
```

Circle

```
record S¹ : U :=
    base : S¹
    loop : Id S¹ base base
    rec : (F : S¹ → U) (b : F base) (l : IdS S¹ F base base loop b b) (x : S¹) → F x
```

Ladder to computable HITs

1. Barendregt. The Lambda Calculus with Types
2. Martin-Löf. Intuitionistic Type Theory
3. Awodey. Category Theory
4. Hermida, Jacobs. Fibrations with indeterminates
5. Jacobs. Categorical Logic
6. Hofmann, Streicher. The groupoid interpretation of type theory.
7. Voevodsky et all. Homotopy Type Theory
8. Huber, Coquand. Cubical Type Theory

Ladder to computable HITs

1. Barendregt. The Lambda Calculus with Types http://5ht.co/pts.pdf
2. Martin-Löf. Intuitionistic Type Theory http://5ht.co/mltt.pdf
3. Awodey. Category Theory http://5ht.co/cat.pdf
4. Hermida, Jacobs. Fibrations with indeterminates http://5ht.co/completeness.pdf
5. Jacobs. Categorical Logic http://5ht.co/fibrations.pdf
6. Streicher. The groupoid interpretation of type theory http://5ht.co/gropuoid.pdf
7. Voevodsky et all. Homotopy Type Theory http://5ht.co/hott.pdf
8. Huber, Coquand. Cubical Type Theory http://5ht.co/cubicaltt.pdf