

MLTT — CT — Proof Theory

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

$\text{Nat} : \mathcal{U} \text{ n}$ — constant functor
 $\text{List} (A : \mathcal{U} \text{ n}) : \mathcal{U} \text{ n}$ — functor
 $\text{List Nat} : \mathcal{U} \text{ n}$ — constant functor

$\text{List } A = 1 \longrightarrow \text{List } A \longrightarrow \text{List } A \longleftarrow A : \mathcal{U}$
 $\text{nil} : 1 \longrightarrow \text{List } A$
 $\text{cons} : A \longrightarrow \text{List } A \longrightarrow \text{List } A$

$\text{Nat} = 1 \longrightarrow \mathbb{N} \longrightarrow \mathbb{N} : \mathcal{U}$
 $\text{zero} : 1 \longrightarrow \mathbb{N}$
 $\text{succ} : \mathbb{N} \longrightarrow \mathbb{N}$

Pure Type System

$U_0 : U_1 : U_2 : U_3 : \dots$

U_0 — propositions

U_1 — sets

U_2 — types

U_3 — sorts

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

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

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

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

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

```

data O1 := Star: nat → O1
        | Var: name → O1
        | App: O1 → O1 → O1
        | Lambda: name → O1 → O1 → O1
        | Arrow: O1 → O1 → O1
        | Pi: name → O1 → O1 → O1.

```

```

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

```

Pi Type

$\forall x: A, B x : U$ — formation rule
 $\lambda x: A, b : B x$ — introduction
 $\text{app } f \ a : B x$ — elimination
 $\text{app } (\lambda o:A, b) \ a = b \ [a/o] : B x$
 — equation

Sigma Types

```
data O2 := Star: nat → O2
        | Var: name → O2
        | App: O2 → O2 → O2
        | Lambda: name → O2 → O2 → O2
        | Arrow: O2 → O2 → O2
        | Pi: name → O2 → O2 → O2
        | Sigma: name → O2 → O2 → O2
        | Pair: O2 → O2 → O2
        | Fst: O2 → O2
        | Snd: O2 → O2.
```

$\Sigma x: A, B x : U$ — formation rule
pair (x : A) (y : B x) — introduction
pr₁ s : A — elimination
pr₂ 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:  $\forall$  (x,y: A) → Id x y → Type  
  Forall (C: Predicate):  $\forall$  (x,y: A) →  $\forall$  (p: Id x y) → C x y p  
   $\Delta$  (C: Predicate):  $\forall$  (x: A) → C x x (refl x)  
  axiom-J (C: Predicate):  $\Delta$  C → Forall C  
  computation (C: Predicate) (t:  $\Delta$  C):  $\forall$  (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 ( $\lambda$  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:  $\forall$  (a: A)  $\rightarrow$  Id a a  $\rightarrow$  Type
```

```
  ForallK (C: PredicateK):  $\forall$  (a: A)  $\rightarrow$   $\forall$  (p: Id a a)  $\rightarrow$  C a p
```

```
   $\Delta$ K (C: PredicateK) :  $\forall$  (a: A)  $\rightarrow$  C a (Id.refl a)
```

```
  axiom-K (C: Predicate):  $\Delta$ K C  $\rightarrow$  ForallK C)
```

```
define Respect (A,B: Type) (C: A  $\rightarrow$  Type) (D: B  $\rightarrow$  Type) (R: A  $\rightarrow$  B  $\rightarrow$  Prop)
```

```
  (Ro:  $\forall$  (x: A) (y: B)  $\rightarrow$  C x  $\rightarrow$  D y  $\rightarrow$  Prop) : ( $\forall$  (x: A)  $\rightarrow$  C x)  $\rightarrow$  ( $\forall$  (x: B)  $\rightarrow$  D x)  $\rightarrow$  Prop
```

```
:=  $\lambda$  (f,g: Type  $\rightarrow$  Type)  $\rightarrow$  ( $\forall$  (x,y: Type)  $\rightarrow$  R x y)  $\rightarrow$  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

Dom₁_o: (x,y: Ob) (f: Hom x y) → (Hom.Equ x y (Comp x x y id f) f)

Cod₁_o: (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)

Subst_o: (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))

Assoc_o: (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:  $\forall$  (x: Carrier) → Equ x x  
  Trans:  $\forall$  (x1,x2,x3: Carrier) → Equ x1 x2 → Equ x2 x3 → Equ x1 x3  
  Sym:  $\forall$  (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 :=
```

```
  frege_S: (p -> q -> r) -> (p -> q) -> p -> r
```

```
  simple_K: p → (q → 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
```

```
data MLTT := Pi:   MLTT  -> MLTT -> MLTT
           | Lam:   Binder -> MLTT -> MLTT
           | App:   MLTT  -> MLTT -> MLTT
           | Sigma: MLTT  -> MLTT -> MLTT
           | Pair:  MLTT  -> MLTT -> MLTT
           | Fst:   MLTT  -> MLTT
           | Snd:   MLTT  -> MLTT
           | Where: MLTT  -> Decls -> MLTT
           | Var:   Ident -> MLTT
           | U:     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.
```

Infinity Language

Homotopy Calculus

Require Import core.

Inductive 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	IO	I1
Line	IntRec	Undef: Loc -> HomotopyCalculus.	

$\text{Id} : (A : \mathcal{U}) (a\ b : A) \rightarrow \mathcal{U}$

$\text{IdP} : (A\ B : \mathcal{U}) \rightarrow \text{Id}\ \mathcal{U}\ A\ B \rightarrow A \rightarrow B \rightarrow \mathcal{U}$

$\text{refl} : (A : \mathcal{U}) (a : A) \rightarrow \text{Id}\ A\ a\ a$

$\text{inh} : \mathcal{U} \rightarrow \mathcal{U}$

$\text{inc} : (A : \mathcal{U}) \rightarrow A \rightarrow \text{inh}\ A$

$\text{squash} : (A : \mathcal{U}) \rightarrow \text{prop}(\text{inh}\ A)$

$\text{inhrec} : (A : \mathcal{U}) (B : \mathcal{U}) (p : \text{prop}\ B) (f : A \rightarrow B) (a : \text{inh}\ A) \rightarrow B$

$\text{contrSingl} : (A : \mathcal{U}) (a\ b : A) (p : \text{Id}\ A\ a\ b) \rightarrow \text{Id}(\text{singl}\ A\ a)(a, \text{refl}\ A\ a)(b, p)$

$\text{equivEq} : (A\ B : \mathcal{U}) (f : A \rightarrow B) (s : (y : B) \rightarrow \text{fiber}\ A\ B\ f\ y)$

$(t : (y : B) \rightarrow (v : \text{fiber}\ A\ B\ f\ y) \rightarrow$

$\text{Id}(\text{fiber}\ A\ B\ f\ y)(s\ y)\ v) \rightarrow \text{Id}\ \mathcal{U}\ A\ B$

$\text{equivEqRef} : (A : \mathcal{U}) \rightarrow (s : (y : A) \rightarrow \text{pathTo}\ A\ y) \rightarrow$

$(t : (y : A) \rightarrow (v : \text{pathTo}\ A\ y) \rightarrow$

$\text{Id}(\text{pathTo}\ A\ y)(s\ y)\ v) \rightarrow$

$\text{Id}(\text{Id}\ \mathcal{U}\ A\ A)(\text{refl}\ \mathcal{U}\ A)(\text{equivEq}\ A\ A(\text{id}\ A)\ s\ t)$

Equiv Squash

Id Inh Inc

Proposition Fibration

Path Singleton

$\text{id} : (A : U) \rightarrow A \rightarrow A$

$\text{id } A \ a = a$

$\text{sld} : (A : U) (a : A) \rightarrow \text{pathTo } A \ a$

$\text{sld } A \ a = (a, \text{refl } A \ a)$

$\text{singl} : (A : U) \rightarrow A \rightarrow U$

$\text{singl } A \ a = \text{Sigma } A \ (\text{Id } A \ a)$

$\text{pathTo} : (A : U) \rightarrow A \rightarrow U$

$\text{pathTo } A = \text{fiber } A \ A \ (\text{id } A)$

$\text{IdS} : (A : U) (F : A \rightarrow U) (a_0 \ a_1 : A) (p : \text{Id } A \ a_0 \ a_1) \rightarrow F \ a_0 \rightarrow F \ a_1 \rightarrow U$

$\text{IdS } A \ F \ a_0 \ a_1 \ p = \text{IdP } (F \ a_0) \ (F \ a_1) \ (\text{mapOnPath } A \ U \ F \ a_0 \ a_1 \ p)$

$\text{prop} : U \rightarrow U$

$\text{prop } A = (a \ b : A) \rightarrow \text{Id } A \ a \ b$

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

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

$\text{fiber} : (A \ B : U) (f : A \rightarrow B) (y : B) \rightarrow U$

$\text{fiber } A \ B \ f \ y = \text{Sigma } A \ (\lambda x \rightarrow \text{Id } B \ (f \ x) \ y)$

Transport App Map

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)

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)

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)

```
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
```

Extensionality

```
record I : U :=
  IO : I
  I1 : I
  line : Id I IO I1
  intrec : (F : I -> U) (s : F IO) (e : F I1) (l : IdS I F IO I1 line s e) (x : I) -> F x
```

```
record S1 : U :=
  base : S1
  loop : Id S1 base base
  S1rec : (F : S1 -> U) (b : F base)
    (l : IdS S1 F base base loop b b) (x : S1) -> F x
```

Interval Circle