Towards an Internalization of the Groupoid Model of Type Theory

Joint work with Matthieu Sozeau (Inria Paris)

Nicolas Tabareau
Ascola, Nantes

Groupoids as enriched sets

Moving away from identity types, we can realize a richer model:

Proof-irrelevance	Irrelevant Equality
Propositional extensionality	Logical equivalence
Functional extensionality	Pointwise equality
Univalence	Isomorphism

Groupoids as enriched sets

- The Groupoid Model – Hofmann and Streicher '94, '98
- UnivalentFoundations–Voevodsky, Awodey, Warren, …
- Takeuti-Gandy for Type Theory (Setoids) – Barras & Coquand, 2013

Goal of our work

- clarify what can be done intentionally with groupoids, clarify the need for identity types
- formalize the interpretation in a modular way
  => replace groupoids by higher dimensional structure in the future
  => require the use of universe polymorphism

Groupoid vs Setoids

The setoid model (at the basis of e.g. OTT), assumes a proof-irrelevant equality:

$$\Sigma A : \texttt{Type}. \Sigma \sim : A \to A \to \texttt{Prop}.$$
$$\Sigma \sim_{equiv} : \forall\ x\ y, \textsf{Equivalence}\ (x \sim y)\ \textsf{eq}.$$
$$\Sigma\ irrel : \forall\ x\ y\ (p\ q : x \sim y), p = q \ldots$$

Groupoid vs Setoids

In Hofmann and Streicher: $[\![T]\!] : GPD$. GPD come with a relevant equality:

$$\Sigma A : \texttt{Type}.\Sigma \sim\, : A \to A \to \texttt{Type}.$$
$$\Sigma \sim_{equiv}\, : \forall\ x\ y, \textsf{Equivalence}\ (x \sim y)\ \textsf{eq} \ldots$$

But morphisms representing identities are still identified up to *propositional* equality ($\textsf{eq} : \Pi A, A \to A \to \texttt{Prop}$).

Intensional vs extensional metatheory

In Hofmann and Streicher, the model is built in an *extensional* metatheory, so eq is reflective there.

In *intensional* type theory, we rather represent groupoids with an additional notion of (2-)equality.

$$\Sigma A : \texttt{Type}.\Sigma \sim_1 : A \to A \to \texttt{Type}.$$
$$\Sigma \sim_2 : \forall \{x\ y\}\ (p\ q : x \sim_1 y), \texttt{Type}.$$
$$\Sigma \sim_{2-equiv} : \forall\ x\ y, \textsf{Equivalence}\ (x \sim_1 y)\ \ \sim_2 \ldots.$$

E.g: $[\![\texttt{Prop}]\!] := (\texttt{Prop}, \mathrm{iff}, \mathrm{irrel}, \ldots)$.

∞-Groupoids

Ideally, we'd like to model ∞-groupoids, avoiding the need for truncation.

The groupoid case limits us:

- we need **functional extensionality**
- we need **identity types** to express it

A truly infinite-dimensional extension would lift these (using e.g, globular sets, operads, cubical sets).

Our groupoid model

Stays agnostic w.r.t. future extension, entirely in categorical language (i.e. Dybjer's CwFs):

- Interprets MLTT with one universe, Π, Σ, Id-types (no type polymorphism) into CIC (actually the ECC fragment + Id-types for truncations).

- Validates extensional equality principles + "Isomorphic types are equal":

$$\frac{\Gamma \vdash i : \texttt{Elt}\ A \equiv \texttt{Elt}\ B}{\Gamma \vdash \texttt{equiv}\ i : \texttt{Id}\ \mathcal{U}\ A\ B}$$

Defining groupoids, internally.

Defining groupoids: Relations

Start with computational, proof-relevant relations:

$$\texttt{Definition}\ \mathsf{HomSet}\ (T : \texttt{Type}) := T \to T \to \texttt{Type}.$$

Using classes and universe polymorphism, we get notations for 1-and 2-dimensional equalities:

$$\texttt{Class}\ \mathsf{HomSet}_1\ T := \{\mathrm{eq}_1 : \mathsf{HomSet}\ T\}.$$
$$\texttt{Infix}\ "\sim_1" := \mathrm{eq}_1\ (\texttt{at level}\ 80).$$

$$\texttt{Class}\ \mathsf{HomSet}_2\ \{T\}\ (Hom : \mathsf{HomSet}\ T) :=$$
$$\{\mathrm{eq}_2 : \forall \{x\ y : T\}, \mathsf{HomSet}\ (Hom\ x\ y)\}.$$
$$\texttt{Infix}\ "\sim_2" := \mathrm{eq}_2\ (\texttt{at level}\ 80).$$

Defining groupoids: Relations

Given a HomSet, we define type classes for equivalences:

```
Class Identity {A} (Hom : HomT A) :=
  identity : ∀ x, Hom x x.

Class Inverse {A} (Hom : HomT A) :=
  inverse : ∀ x y:A, Hom x y → Hom y x.

Class Composition {A} (Hom : HomT A) :=
  composition : ∀ {x y z:A}, Hom x y → Hom y z → Hom x z.
```

Pre-categories

In a PreCategory, coherences are given up-to $\sim_2$.

```
Class PreCategory T := { Hom₁ :> HomSet₁ T; Hom₂ :> HomSet₂ eq₁;
      Id :> Identity eq₁; Comp :> Composition eq₁;
      Equivalence₂ :> ∀ x y, (Equivalence (eq₂ (x:=x) (y:=y)));
      id_R : ∀ x y (f : x ~₁ y), f ∘ identity x ~₂ f ;
      id_L : ∀ x y (f : x ~₁ y), identity y ∘ f ~₂ f ;
      assoc : ∀ x y z w (f: x ~₁ y) (g: y ~₁ z) (h: z ~₁ w),
                  (h ∘ g) ∘ f ~₂ h ∘ (g ∘ f);
      comp : ∀ x y z (f f': x ~₁ y) (g g': y ~₁ z),
                  f ~₂ f' → g ~₂ g' → g ∘ f ~₂ g' ∘ f' }.
```

Pre-groupoids

A PreGroupoid is a PreCategory where all 1-Homs are invertible and subject to additional compatibility laws for inverses.

$$\texttt{Class}\ \mathsf{PreGroupoid}\ T := \{\ \mathrm{C} :> \mathsf{PreCategory}\ T\ ;\ \mathrm{Inv} :> \mathsf{Inverse}\ \mathrm{eq}_1\ ;$$
$$\mathrm{inv}_R : \forall\ x\ y\ (f\!: x \sim_1 y),\ f \circ f^{-1} \sim_2 \mathrm{identity}\ y\ ;$$
$$\mathrm{inv}_L : \forall\ x\ y\ (f\!: x \sim_1 y),\ f^{-1} \circ f \sim_2 \mathrm{identity}\ x\ ;$$
$$\mathrm{inv} : \forall\ x\ y\ (f\ f'\!: x \sim_1 y),\ f \sim_2 f' \rightarrow f^{-1} \sim_2 f'^{-1} \}.$$

Groupoids and Contractibility

Groupoids are then PreGroupoids where

equality at dimension 2 is irrelevant

This irrelevance is defined using a notion of contractibility expressed with (relevant) Identity Types.

```
Class Contr (A : Type) := {
  center : A ;
  contr : ∀ y : A, center = y}.
```

Groupoids and Contractibility

By analogy to homotopy type theory, we note IsType_1 the property of being a groupoid.

$$\texttt{Class}\ \mathsf{IsType}_1\ T := \{\ \mathrm{G} :> \mathsf{PreGroupoid}\ T\ ;\ \mathrm{is\_Trunc\_2} : \forall\ (x\ y : T)\ (e\ e' : x \sim_1 y)\ (E\ E' : e \sim_2 e'),\ \mathsf{Contr}\ (E = E')\}.$$

In the same way, we define IsType_0 when equality is irrelevant at dimension 1.

$$\texttt{Class}\ \mathsf{IsType}_0\ T := \{\ \mathrm{S} :> \mathsf{IsType}_1\ T\ ;\ \mathrm{is\_Trunc\_1} : \forall\ (x\ y : T)\ (e\ e' : x \sim_1 y)\ ,\ \mathsf{Contr}\ (e = e')\}.$$

Functors and natural transformations.

Functors and natural transformations

Groupoid morphisms are functors:

```
Class Functor {T U : Type₁} (f : [T] → [U]) : Type :=
{ map : ∀ {x y}, x ~₁ y → f x ~₁ f y ;
  map_comp : ∀ {x y z} (e:x ~₁ y) (e':y ~₁ z),
    map (e' ∘ e) ~₂ map e' ∘ map e ;
  map₂ : ∀ {x y:[T]} {e e' : x ~₁ y}, (e ~₂ e') → map e ~₂ map e' }.
Definition Fun_Type (T U : Type₁) := {f : [T] → [U] & Functor f}.
```

$T \longrightarrow U$ are functors from T to U

$M \star N$ is the application of the functor M to N

Functors and natural transformations

need compatibility at level 2

Groupoid morphisms are functors:

```
Class Functor {T U : Type₁} (f : [T] → [U]) : Type :=
{ map : ∀ {x y}, x ∼₁ y → f x ∼₁ f y ;
  map_comp : ∀ {x y z} (e:x ∼₁ y) (e':y ∼₁ z),
    map (e' ∘ e) ∼₂ map e' ∘ map e ;
  map₂ : ∀ {x y:[T]} {e e' : x ∼₁ y}, (e ∼₂ e') → map e ∼₂ map e' }.
Definition Fun_Type (T U : Type₁) := {f : [T] → [U] & Functor f}.
```

$T \longrightarrow U$ are functors from T to U

$M \star N$ is the application of the functor M to N

Natural transformations

Equivalence between functors is given by natural transformations

$$\texttt{Class}\ \textsf{NaturalTrans}\ T\ U\ \{f\ g : T \longrightarrow U\}\ (\alpha : \forall\ t : [T],\ f \star t \sim_1 g \star t)$$
$$:= \alpha_{\textsf{map}} : \forall\ \{t\ t'\}\ (e : t \sim_1 t'),\ \alpha\ t' \circ \texttt{map}\ f\ e \sim_2 \texttt{map}\ g\ e \circ \alpha\ t.$$

$$\texttt{Definition}\ \mathrm{nat\_trans}\ T\ U : \textsf{HomSet}\ (T \longrightarrow U)$$
$$:= \lambda\ f\ g,\ \{\alpha : \forall\ t : [T],\ f \star t \sim_1 g \star t\ \&\ \textsf{NaturalTrans}\ \alpha\}.$$

The function space groupoid structure

Functors, natural transformations and modifications form a PreGroupoid.

It requires functional extensionality to prove the truncation property of Groupoids.

The (Pre-)Groupoid of Groupoids.

Homotopy equivalences

Equivalence between groupoids is given by equivalences.

$$
\begin{aligned}
&\texttt{Class}\ \mathsf{Iso\_struct}\ T\ U\ (f : [T \longrightarrow U]) := \\
&\{\ \text{adjoint} : [U \longrightarrow T]\ ; \\
&\quad \text{section} : f \circ \text{adjoint} \sim_2 \text{identity}\ U\ ; \\
&\quad \text{retraction} : \text{adjoint} \circ f \sim_2 \text{identity}\ T \}.
\end{aligned}
$$

Homotopy equivalences

Actually, we use the triangle identity to get adjoint equivalences (it turns it into an hProp).

$$\texttt{Class}\ \mathsf{Equiv\_struct}\ T\ U\ (f : T \longrightarrow U) :=$$
$$\{\ \mathrm{iso} : \mathsf{Iso\_struct}\ f;$$
$$\quad \mathrm{triangle} : \forall\ t,\ \mathrm{section} \star (f \star t) \sim_2 \texttt{map}\ f\ (\mathrm{retraction} \star t)\}.$$

The (Pre-)Groupoid of Groupoids

We can define the PreGroupoid of Groupoids and homotopy equivalences.

It does not form a Groupoid.

But Setoids do form a groupoid.

$$\texttt{Definition}\ \mathsf{Type}_0^1 : \mathsf{Type}_1 := (\mathsf{Type}_0\ ;\ \mathsf{Equiv}_{\mathsf{Type}_0}).$$

Remark: Groupoids appear both in the type and the term => requires universe polymorphism.

Rewriting.

Rewriting: Transport

Transport is given by functoriality:

$$\texttt{Definition}\ \mathrm{transport}\ A\ (F\!:\![A \longrightarrow \mathsf{Type}^1_1])\ \{x\ y\!:\![A]\}\ (e\!:\!x \sim_1 y)$$
$$:\ (F \star x) \longrightarrow (F \star y) := [\mathtt{map}\ F\ e].$$

The equational theory is derivable from the groupoid laws, e.g.:

$$\texttt{Definition}\ \mathsf{transport}_{\mathsf{eq}}\ A\ (F\!:\![A \longrightarrow \mathsf{Type}^1_1])\ \{x\ y\!:\![A]\}$$
$$\{e\ e'\!:\!x \sim_1 y\}\ (H\!:\!e \sim_2 e')$$
$$:\ \mathrm{transport}\ F\ e \sim_1 \mathrm{transport}\ F\ e' := [\mathrm{map}_2\ F\ H].$$

Dependent Functions.

Dependent Functors

As for functions, dependent functions are interpreted as functors, but of a dependent kind.

$$
\begin{array}{l}
\texttt{Class}\ \mathsf{Functor}^{\Pi}\ T\ (U : [T \longrightarrow \mathsf{Type}_1^1])\ (f : \forall\ t,\ [U \star t]) : \texttt{Type} := \{ \\
\quad \mathrm{map}^{\Pi} : \forall\ \{x\ y\}\ (e\!: x \sim_1 y),\ \mathrm{transport}\ U\ e \star (f\ x) \sim_1 f\ y\ ; \\
\quad \mathrm{map}^{\Pi}_{\mathsf{id}} : \forall\ x,\ \mathrm{map}^{\Pi}\ (\mathrm{identity}\ x) \sim_2 \mathsf{transport_{id}}\ U \star (f\ x); \\
\quad \mathrm{map}^{\Pi}_{\mathsf{comp}} : \forall\ x\ y\ z\ (e : x \sim_1 y)\ (e' : y \sim_1 z), \\
\quad\quad \mathrm{map}^{\Pi}\ (e' \circ e) \sim_2 \mathrm{map}^{\Pi}\ e' \circ \mathsf{transport_{map}}\ U\ \_\ (\mathrm{map}^{\Pi}\ e) \circ \\
\quad\quad\quad\quad\quad\quad (\mathsf{transport_{comp}}\ U\ e\ e' \star \_); \\
\quad \mathrm{map}^{\Pi}_2 : \forall\ x\ y\ (e\ e'\!: x \sim_1 y)\ (H\!: e \sim_2 e'), \\
\quad\quad \mathrm{map}^{\Pi}\ e \sim_2 \mathrm{map}^{\Pi}\ e' \circ (\mathsf{transport_{eq}}\ U\ H \star (f\ x))\}.
\end{array}
$$

Dependent Functors

As for functions, dependent functions are interpreted as functors, but of a dependent kind.

$$
\begin{array}{l}
\texttt{Class}\ \mathsf{Functor}^{\Pi}\ T\ (U : [T \longrightarrow \mathsf{Type}^1_1])\ (f : \forall\ t,\ [U \star t]) : \mathsf{Type} := \{ \\
\quad \mathrm{map}^{\Pi} : \forall\ \{x\ y\}\ (e : x \sim_1 y),\ \mathsf{transport}\ U\ e \star (f\ x) \sim_1 f\ y\ ; \\
\quad \mathrm{map}^{\Pi}_{\mathsf{id}} : \forall\ x,\ \mathrm{map}^{\Pi}\ (\mathrm{identity}\ x) \sim_2 \mathsf{transport}_{\mathsf{id}}\ U \star (f\ x); \\
\quad \mathrm{map}^{\Pi}_{\mathsf{comp}} : \forall\ x\ y\ z\ (e : x \sim_1 y)\ (e' : y \sim_1 z), \\
\quad\quad \mathrm{map}^{\Pi}\ (e' \circ e) \sim_2 \mathrm{map}^{\Pi}\ e' \circ \mathsf{transport}_{\mathsf{map}}\ U\ \_\ (\mathrm{map}^{\Pi}\ e) \circ \\
\quad\quad\quad\quad\quad\quad\quad\quad (\mathsf{transport}_{\mathsf{comp}}\ U\ e\ e' \star \_); \\
\quad \mathrm{map}^{\Pi}_{2} : \forall\ x\ y\ (e\ e' : x \sim_1 y)\ (H : e \sim_2 e'), \\
\quad\quad \mathrm{map}^{\Pi}\ e \sim_2 \mathrm{map}^{\Pi}\ e' \circ (\mathsf{transport}_{\mathsf{eq}}\ U\ H \star (f\ x))\}.
\end{array}
$$

provides a dependent version of transport

Dependent Natural Transformations

Equality between **dependent** functors is given by **dependent** natural transformations.

$$\texttt{Class}\ \mathsf{NaturalTrans}^{\Pi}\ T\ (U\!:\![T \longrightarrow \mathsf{Type}^1_1])\ \{f\ g\!:\ \Pi_T\ U\}$$
$$(\alpha : \forall\ t,\ f \star t \sim_1 g \star t) :=$$
$$\alpha_{\mathsf{map}^{\Pi}} : \forall\ \{t\ t'\}\ e,\ \alpha\ t' \circ \mathrm{map}^{\Pi}\ f\ e \sim_2 \mathrm{map}^{\Pi}\ g\ e \circ \mathsf{transport}_{\mathsf{map}}\ U\ e\ (\alpha\ t).$$

$$\texttt{Definition}\ \mathsf{nat\_trans}^{\Pi}\ T\ (U\!:\![T \longrightarrow \mathsf{Type}^1_1])\ (f\ g\!:\ \Pi_T\ U)$$
$$:= \{\alpha : \forall\ t : [T],\ f \star t \sim_1 g \star t\ \&\ \mathsf{NaturalTrans}^{\Pi}\ \alpha\}.$$

Dependent Natural Transformations

Equality between **dependent** functors is given by **dependent** natural transformations.

$$\texttt{Class}\ \mathsf{NaturalTrans}^{\Pi}\ T\ (U{:}[T \longrightarrow \mathsf{Type}_1^1])\ \{f\ g{:}\ \Pi_{\mathrm{T}}\ U\}$$
$$(\alpha : \forall\ t,\ f \star t \sim_1 g \star t) :=$$
$$\alpha_{\mathsf{map}^{\Pi}} : \forall\ \{t\ t'\}\ e,\ \alpha\ t' \circ \mathrm{map}^{\Pi}\ f\ e \sim_2 \mathrm{map}^{\Pi}\ g\ e \circ \mathsf{transport}_{\mathsf{map}}\ U\ e\ (\alpha\ t).$$

$$\texttt{Definition}\ \mathsf{nat\_trans}^{\Pi}\ T\ (U{:}[T \longrightarrow \mathsf{Type}_1^1])\ (f\ g{:}\ \Pi_{\mathrm{T}}\ U)$$
$$:= \{\alpha : \forall\ t : [T],\ f \star t \sim_1 g \star t\ \&\ \mathsf{NaturalTrans}^{\Pi}\ \alpha\}.$$

Dependent functors form a Groupoid.

Dependent Sums.

Dependent Sums

In the interpretation of Σ types, we pay for the fact that we are missing the 2-dimensional nature of groupoids.

We must restrict to codomains in setoids.

$$\texttt{Definition}\ \Sigma_{\mathrm{T}}\ T\ (U : [T \longrightarrow \mathsf{Type}_0^1]) := \{t : [T]\ \&\ [U \star t]\}.$$

The interpretation of Type Theory.

Takeuti-Gandy style interpreation

Following Dybjer, Hofmann&Streicher, Coquand et al., we interpret:

- Context Γ as a Groupoid
- Type $\Gamma \vdash A$ as a functor from Γ to the Groupoid of setoids
- Context extension $\Gamma, x : A \vdash$ is given by dependent sums
- Term $\Gamma \vdash x : A$ as a functor from Γ to A

Takeuti-Gandy style interpreation

Following Dybjer, Hofmann&Streicher, Coquand et al., we interpret:

- Context Γ as a ∞-Groupoid
- Type $\Gamma \vdash A$ as a functor from Γ to the ∞-Groupoid of ∞-Groupoid
- Context extension $\Gamma, x : A \vdash$ is given by dependent sums
- Term $\Gamma \vdash x : A$ as a ∞-functor from Γ to A

2 views on Dependent Types

A dependent type $\Gamma, x : A \vdash B$ is interpreted in two equivalent ways:

- As a functor from ΣA to setoids
- As a type family over A (corresponding to a family of sets in constructive mathematics). A type family can be seen as a fibration from B to A.

$$\texttt{Definition}\ \mathrm{TypFam}\ \{\Gamma : \mathrm{Context}\}\ (A\colon \mathrm{Typ}\ \Gamma) := [\Pi\ (\lambda\ \gamma,\ (A \star \gamma)\ \restriction_s \longrightarrow \mathsf{Type}^1_0;\ \mathsf{TypFam}_{\mathsf{comp}}\ \_)].$$

2 views on Dependent Types

Those 2 views can be related using a dependent closure at the level of types.

In the interpretation of typing judgments, this connection is used to switch between the fibration and the morphism points of view.

The translation

Using those notions, we can define the translation of TT

$$\texttt{Definition}\ \mathrm{Var}\ \{\Gamma\}\ (A{:}\mathrm{Typ}\ \Gamma) : \mathsf{Tm}\ \uparrow A := (\lambda\ t,\ \pi_2\ t;\ \mathsf{Var}_{\mathsf{comp}}\ A).$$

$$\texttt{Definition}\ \mathrm{Prod}\ \{\Gamma\}\ (A{:}\mathrm{Typ}\ \Gamma)\ (F{:}\mathrm{TypFam}\ A)$$
$$: \mathrm{Typ}\ \Gamma := (\lambda\ s,\ \Pi_0\ (F \star s);\ \mathsf{Prod}_{\mathsf{comp}}\ A\ F).$$

$$\texttt{Definition}\ \mathrm{App}\ \{\Gamma\}\ \{A{:}\mathrm{Typ}\ \Gamma\}\ \{F{:}\mathrm{TypFam}\ A\}$$
$$(c{:}\mathsf{Tm}\ (\mathrm{Prod}\ F))\ (a{:}\mathsf{Tm}\ A) : \mathsf{Tm}\ (F\ \{\{a\}\}) :=$$
$$(\lambda\ s,\ (c \star s) \star (a \star s);\ \mathsf{App}_{\mathsf{comp}}\ c\ a).$$

$$\texttt{Definition}\ \mathrm{Lam}\ \{\Gamma\}\ \{A{:}\mathrm{Typ}\ \Gamma\}\ \{B{:}\mathrm{TypDep}\ A\}\ (b{:}\mathsf{Tm}\ B)$$
$$: \mathsf{Tm}\ (\mathrm{Prod}\ (\Lambda\ B)) := (\lambda\ \gamma,\ (\lambda\ t,\ b \star (\gamma\ ;\ t)\ ;\ \_);\ \mathsf{Lam}_{\mathsf{comp}}\ b).$$

$$\texttt{Definition}\ \mathrm{Sigma}\ \{\Gamma\}\ (A{:}\mathrm{Typ}\ \Gamma)\ (F{:}\mathrm{TypFam}\ A)$$
$$: \mathrm{Typ}\ \Gamma := (\lambda\ \gamma{:}\ [\Gamma],\ \Sigma\ (F \star \gamma);\ \mathsf{Sigma}_{\mathsf{comp}}\ A\ F).$$

$$\texttt{Definition}\ \mathrm{Beta}\ \{\Gamma\}\ \{A{:}\mathrm{Typ}\ \Gamma\}\ \{F{:}\mathrm{TypDep}\ A\}\ (b{:}\mathsf{Tm}\ F)\ (a{:}\mathsf{Tm}\ A)$$
$$: [\mathrm{Lam}\ b \star a] = [b \circ \mathrm{SubExtId}\ a] := \mathsf{eq\_refl}\ \_.$$

The translation

Using those notions, we can define the translation of TT

```
Definition Var {Γ} (A:Typ Γ) : Tm ↑A := (λ t, π₂ t; Var_comp A).

Definition Prod {Γ} (A:Typ Γ) (F:TypFam A)
  : Typ Γ := ...

Definition App {Γ} {A:Typ Γ} {F:TypFam A}
  (c:Tm (Prod F)) (a:Tm A) : Tm (F {{a}}) :=
  ...

Definition Lam {Γ} {...} (b:Tm B)
  : Tm (Prod (Λ B)) := (λ γ, (λ t, b ⋆ (γ ; t) ; _); Lam_comp b).

Definition Sigma {Γ} (A:Typ Γ) (F:TypFam A)
  : Typ Γ := (λ γ: [Γ], Σ (F ⋆ γ); Sigma_comp A F).

Definition Beta {Γ} {A:Typ Γ} {F:TypDep A} (b:Tm F) (a:Tm A)
  : [Lam b ⋆ a] = [b ∘ SubExtId a] := eq_refl _.
```

Not Completely formalised

Need more automated reasoning using relevant rewriting (see next talk!)

Identity Types

The meaning of the identity types is given
by induction on types instead of by an inductive type.

```
Definition Id {Γ} (A: Typ Γ) (a b : Tm A)
  : Typ Γ := (λ γ, (a ⋆ γ ∼₁ b ⋆ γ ; _); Id_comp A a b).
```

Identity Types

The meaning of the identity types is given by induction on types instead of by an inductive type.

$$\texttt{Definition}\ \mathrm{Id}\ \{\Gamma\}\ (A\colon \mathrm{Typ}\ \Gamma)\ (a\ b : \mathsf{Tm}\ A)$$
$$: \mathrm{Typ}\ \Gamma := (\lambda\ \gamma,\ (a \star \gamma \sim_1 b \star \gamma\ ;\ \_)\ ;\ \mathsf{Id}_{\mathsf{comp}}\ A\ a\ b).$$

We can interpret the J eliminator of MLTT on Id using functoriality of P and products.

The J equality rule holds up to $\sim_2$ in the model.

Univalent Type Theory

As equality between setoids is given by (adjoint) equivalence, we get a type theory with a univalent universe.

The precise formulation is still work in progress.

Doggy Bag

- Groupoids can be internalised but this requires functional extensionality and the use of identity types for contractibility.
- We have a (partially) formalised interpretation of a type theory with a univalent universe.
- Should scale to higher-order models (ie. cubical sets)