Polynomial Functors in Lean 4

SMS Spring Meeting on Formalization and Proof Assistants

Brig, March 2026

Sina Hazratpour

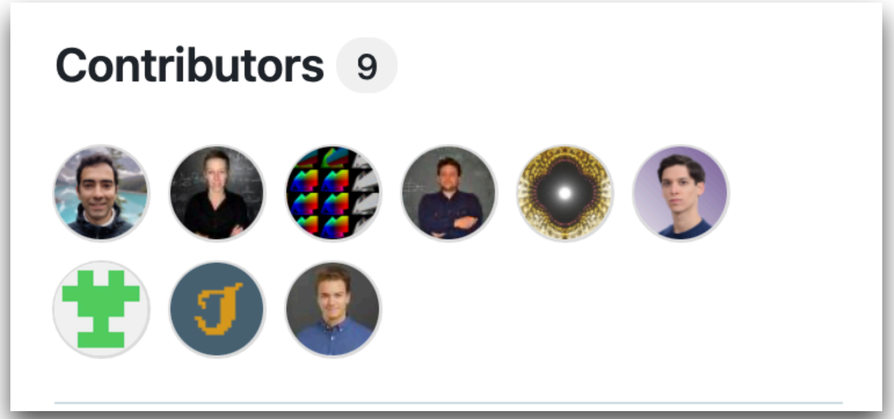
https://sinhp.github.io/Poly/

Why Formalize?

Polynomial functors are one of the most useful tools in category theory.

- Functional Programming & Type Theory:
  Inductive Data Types (W-Types), Generic Programming
- Semantics of Dependent Type Theories
- Combinatorics Species
- Categorical Dynamical Systems & Wiring Diagrams
- Higher Algebra: Operads, Goodwillie Calculus of Functors

- In a downstream Lean project (HoTTLean) we extensively rely on the Lean formalization of polynomial functors.



Polynomials from Algebra

- In Lean4, A (univariate) polynomial with coefficients in a semiring is defined as the monoid algebra over the semiring.

```
structure Polynomial (R : Type*) [Semiring R] where ofFinsupp ::
  /-- The coefficients `ℕ →₀ R` of a polynomial in `R[X]`. -/
  toFinsupp : AddMonoidAlgebra R ℕ
```

- However, we might like to represent them by a sparse data structure consisting of a list of pairs (index, coefficient), sorted according to index.

```
structure UvPoly (R : Type) [CommRing R] where
  coeffs : List (ℕ × R)
  sorted : coeffs.Sorted (·.1 > ·.1)
  nonzero : ∀ x ∈ coeffs, x.2 ≠ 0
```

Polynomial Functors

Polynomial Functors can be seen as a categorification of polynomials with natural number coefficients.

$$P(x) = \sum_{i \in \mathbb{N}} x^{e_i} \quad \xRightarrow{\text{categorification}} \quad P(X) = \sum_{i \in I} X^{E_i}$$

Functoriality

$$X \xrightarrow{\quad f \quad} Y$$

$$P(X) \xrightarrow{\quad P(f) \quad} P(Y)$$

$$\sum_{i \in I} X^{E_i} \xrightarrow{\quad P(f) \quad} \sum_{i \in I} Y^{E_i}$$

$$(i,\ k) \longmapsto (i,\ f \circ k)$$

Polynomials in the Category of Types (~300 LoC)

The data specifying a polynomial functor in the category of types consists of

- A type of positions (aka shapes, aka operation)
- A type of directions (aka arities) for each position

```
structure Poly where
  Pos : Type u
  Dir : Pos → Type v
```

A monomial $\alpha * X^{\beta}$

A linear monomial $\alpha * X$

The constant polynomial $\beta * X^0$

```
def monomial (α β : Type*) : Poly := ⟨α, fun _ ⇒ β⟩

def linear (α : Type*) : Poly := ⟨α, fun _ ⇒ PUnit⟩

def constant (β : Type*) : Poly := ⟨β, fun _ ⇒ PEmpty⟩
```

The Associated Polynomial Functor

```
variable (P : Poly.{u, v})

/-- The object part of the functor associated to a polynomial `P`. -/
@[coe]
def Obj (X : Type v) :=
  Σ i : P.Pos, P.Dir i → X


instance : CoeFun Poly.{u, v} (fun _ ⇒ Type v → Type (max u v)) where
  coe := Obj


/-- The morphism part of the functor associated to a polynomial `P`. -/
def map {X Y : Type v} (f : X → Y) : P X → P Y :=
  fun ⟨i, k⟩ ⇒ ⟨i, f ∘ k⟩


def functor : Type v ⥤ Type (max u v) where
  obj X := P X
  map {X Y} f := P.map f
```

Classic Examples Formalized

Polynomial	Functor	Algebra	Initial Algebra
$\mathsf{Pos} = \{z, s\}$ $\mathsf{Dir}(z) = \emptyset$ $\mathsf{Dir}(s) = \{*\}$	$1 + X$	$1 + X \to X$	$\mathbb{N}$
$\mathsf{Pos} = \{l, n\}$ $\mathsf{Dir}(l) = \emptyset$ $\mathsf{Dir}(n) = \{L, R\}$	$1 + X^2$	$1 + X^2 \to X$	Binary Trees
$\mathsf{Pos} = \{nil\} \cup A$ $\mathsf{Dir}(nil) = \emptyset$ $\mathsf{Dir}(a) = \{*\}$	$1 + A \times X$	$1 + A \times X \to X$	$\mathsf{List}(A)$
$\mathsf{Pos} = \mathbb{N}$ $\mathsf{Dir}(n) = \mathrm{Fin}\, n$	$\sum_{n \in \mathbb{N}} X^n$	$\sum_{n \in \mathbb{N}} X^n \to X$	Rose Trees

Composition of Polynomials

$$P(X) = \sum_{i:I} X^{E_i} \qquad Q(X) = \sum_{j:J} X^{D_j}$$

$$\begin{aligned}
(P \triangleleft Q)(X) &= P(Q(X)) \\
&= \sum_{i:I} \left(\sum_{j:J} X^{D_j} \right)^{E_i} \\
&\cong \sum_{i:I} \sum_{f:E_i \to J} \prod_{e:E_i} X^{D_{f(e)}} \\
&\cong \sum_{(i,f):\sum_{i:I} J^{E_i}} X^{\sum_{e:E_i} D_{f(e)}}
\end{aligned}$$

```
def comp (Q P : Poly) : Poly :=
  ⟨Σ b : P.Pos, P.Dir b → Q.Pos,
    fun ⟨b, c⟩ ↦ Σ e : P.Dir b, Q.Dir (c e)⟩

def compIso :
    (Q.comp P).functor ≅ Q.functor ⋙ P.functor where
  hom := {app := fun X ⇒ fun ⟨b,e⟩ ⇒
    ⟨ b.1, fun x' ⇒ ⟨ b.2 x', fun b' ⇒ e ⟨x',b'⟩ ⟩⟩}
  inv := {app X := comp.mk P Q}
```

Generalizing to Other Categories

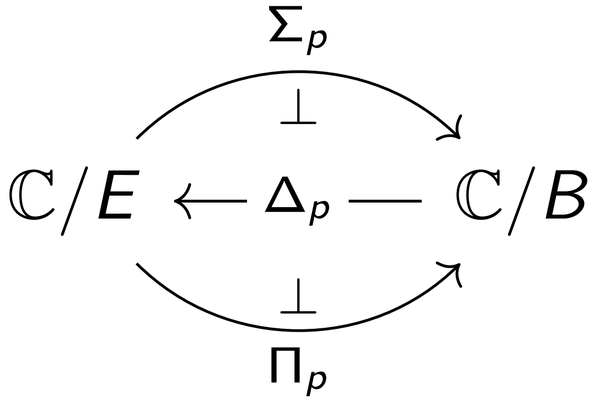
Polynomial functors show up in categories other than the category of types:

- Category of Contexts of Dependent Type Theories
- Presheaf Categories
- Locally Cartesian Closed Categories (LCCCs)
- Category of Groupoids (and other non-LCCCs)

Exponentiable Morphisms

Let $\mathbb{C}$ be a category with finite limits.

Let $p : E \to B$ be an exponentiable morphism in $\mathbb{C}$. Thus:

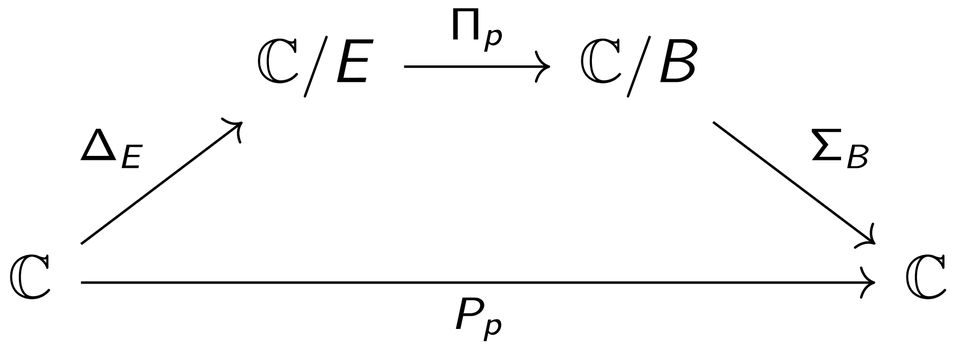


When $B = 1$ is the terminal object, we write

$$\Sigma_E \vdash \Delta_E \vdash \Pi_E$$

Polynomial Functor of Exponentiable Morphisms

The polynomial endofunctor $P_p : \mathbb{C} \to \mathbb{C}$ associated with the morphism $p\colon E \to B$ is the composite



In the internal language of $\mathbb{C}$,

$$P_p(X) = \sum_{b:B} X^{E(b)}$$

Implementation Notes

- We want good computational properties for polynomial functors.
- The API of pullbacks and finite limits in Mathlib are non-constructive: it uses the axiom of global choice to define limits, it gives the user no control over the computation.
- We design a new API for **Chosen Pullbacks** that computes well. Chosen Pullbacks in the category of types and presheaf categories agree with their concrete constructions (up to definitional equality).
- From chosen pullbacks in a category we obtain **cartesian monoidal structures on slices.**

Implementation Notes

- Many theorems about polynomial functors involves proofs which are usually obtained by chaining the universal properties of various pullbacks and push-forwards — this quickly leads to long and uninteresting proofs.
- To address this, we refactored such formal proofs through the 2-categorical calculus mates (j.w.w. Emily Riehl).
- The advantage: calculus mates is purely equational and involves no universal properties. Proving certain transformations are isomorphisms is reduced to checking equality of horizontal and vertical pasting squares (up to defeq).

Chosen Pullbacks (~1000 LoC)

```
class ChosenPullbacksAlong {Y X : C} (f : Y ⟶ X) where
  pullback : Over X ⥤ Over Y
  mapPullbackAdj (f) : Over.map f ⊣ pullback

variable (C) in
abbrev ChosenPullbacks := ∏ {X Y : C} (f : Y ⟶ X), ChosenPullbacksAlong f
```

```
abbrev fst' : (Over.map g).obj ((pullback g).obj (Over.mk f)) ⟶ Over.mk f :=
  (mapPullbackAdj g).counit.app ◁ Over.mk f
```

```
abbrev fst : pullbackObj f g ⟶ Y := fst' f g ▷.left

abbrev snd : pullbackObj f g ⟶ Z := (pullback g).obj (Over.mk f) ▷.hom
```

```
def pullbackCone : PullbackCone f g :=
  PullbackCone.mk (fst f g) (snd f g) (by rw [condition])

/-- The canonical pullback cone is a limit cone. -/
def isLimitPullbackCone :
    IsLimit (pullbackCone f g) :=
  PullbackCone.IsLimit.mk condition (fun s ↦ lift s.fst s.snd s.condition)
    (by cat_disch) (by cat_disch) (by cat_disch)
```

Chosen Pullbacks => CMS on Slices

- If a category comes equipped with chosen pullbacks along all its morphisms, then there is an induced cartesian monoidal structure on all its slice categories.

```
/-- A computable instance of `CartesianMonoidalCategory` for `Over X` when `C` has
chosen pullbacks. Contrast this with the noncomputable instance provided by
`CategoryTheory.Over.cartesianMonoidalCategory`.
-/
def cartesianMonoidalCategoryOver [ChosenPullbacks C] (X : C) :
    CartesianMonoidalCategory (Over X) :=
  ofChosenFiniteProducts (C := Over X)
    ⟨Limits.asEmptyCone (Over.mk (𝟙 X)), Limits.IsTerminal.ofUniqueHom (fun Y ↦ Over.homMk Y.hom)
      fun Y m ↦ Over.OverMorphism.ext (by simpa using m.w)⟩
    (fun Y Z ↦ ⟨ _ , binaryFanIsBinaryProduct Y Z⟩)
```

- Moreover, all the pullback functors are strong monoidal.

Exponentiable Morphisms & Pushforwards (~300 LoC)

```
class ExponentiableMorphism {E B : C} (p : E → B) [ChosenPullbacksAlong p] where
  pushforward : Over E ⥤ Over B
  pullbackPushforwardAdj (p) : pullback p ⊣ pushforward
```

```
def comp {I J K : C} (f : I → J) (g : J → K)
    [ChosenPullbacksAlong f] [ChosenPullbacksAlong g] [ChosenPullbacksAlong (f ≫ g)]
    [ExponentiableMorphism f] [ExponentiableMorphism g] :
    ExponentiableMorphism (f ≫ g) :=
  ⟨pushforward f ⋙ pushforward g,
    ofNatIsoLeft (pullbackPushforwardAdj g ▷.comp ◁ pullbackPushforwardAdj f)
    (pullbackComp f g).symm⟩

def pushforwardComp {I J K : C} (f : I → J) (g : J → K)
    [ChosenPullbacksAlong f] [ChosenPullbacksAlong g] [ChosenPullbacksAlong (f ≫ g)]
    [ExponentiableMorphism f] [ExponentiableMorphism g] [ExponentiableMorphism (f ≫ g)] :
    pushforward (C:= C) (f ≫ g) ≅ pushforward f ⋙ pushforward g :=
  Adjunction.rightAdjointUniq (pullbackPushforwardAdj (f ≫ g))
    ((comp f g).pullbackPushforwardAdj)
```

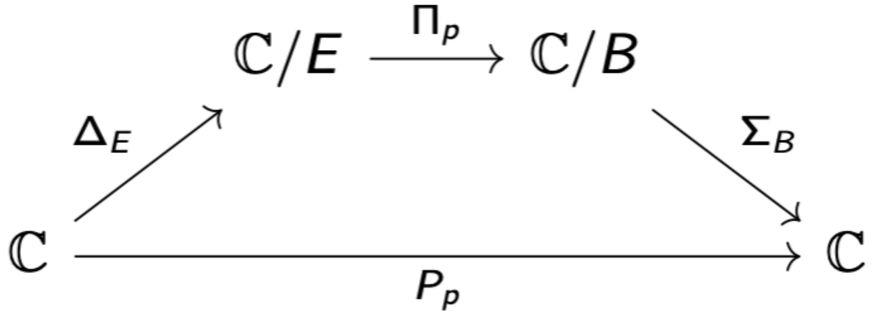
Locally Cartesian Closed Categories (~500 LoC)

Home of Polynomial Functors

- A category is defined to be locally cartesian closed (LCC) if all of its slices are cartesian closed, that is all objects are exponentiable.

- Theorem: A category is LCC iff its has chosen pullbacks along all its morphisms, an all of its morphisms are exponentiable.

- Every morphism is an LCC category has an associated polynomial functor.



Implementation

```
class LocallyCartesianClosed [ChosenPullbacks C] extends ChosenPushforwards C where
  /-- every slice category `Over I` is cartesian closed. This is filled in by default. -/
  cartesianClosedOver : Π (I : C), CartesianClosed (Over I) := cartesianClosedOver
```

```
instance ofChosenPushforwards [ChosenPushforwards C] : LocallyCartesianClosed C where

instance ofCartesianClosedOver [Π (I : C), CartesianClosed (Over I)] :
  LocallyCartesianClosed C where
```

Advantage:

- when using a LCC structure, both the pushforward functor and the cartesian closedness of slices are automatically available,

- whereas for proving locally cartesian closedness proving only one of these two conditions is sufficient.

Examples of LCCCs

1. Category of Types

For any $f : I \to J$, base change $\Delta_f : \mathbf{Type}_{/J} \to \mathbf{Type}_{/I}$ admits both adjoints ($\Sigma_f \dashv \Delta_f \dashv \Pi_f$):

$$\Sigma_f(A \xrightarrow{a} I) = (A \xrightarrow{f \circ a} J)$$

$$\Delta_f(B \xrightarrow{b} J)(i) = B(f(i))$$

$$\Pi_f(A \xrightarrow{a} I)(j) = \prod_{i \in f^{-1}(j)} A(i)$$

2. Category of Presheaves

For any $X \in \mathbf{PSh}(\mathbb{C})$, the slice category satisfies the equivalence:

$$\mathbf{PSh}(\mathbb{C})_{/X} \simeq \mathbf{PSh}\left(\int_{\mathbb{C}} X\right)$$

$\int_{\mathbb{C}} X$ is a small category (Category of Elements).

PSh(small category) is strictly cartesian closed.

$\implies$ $\mathbf{PSh}(\mathbb{C})_{/X}$ is cartesian closed.

$\implies$ $\mathbf{PSh}(\mathbb{C})$ is locally cartesian closed.

Univariate Polynomials in General Categories

```
structure UvPoly (E B : C) where
  (p : E → B)
  (pb : ChosenPullbacksAlong p := by infer_instance)
  (exp : ExponentiableMorphism p := by infer_instance)

def functor (P : UvPoly E B) : C ⥤ C :=
  star E ⋙ pushforward P.p ⋙ Over.forget B
```

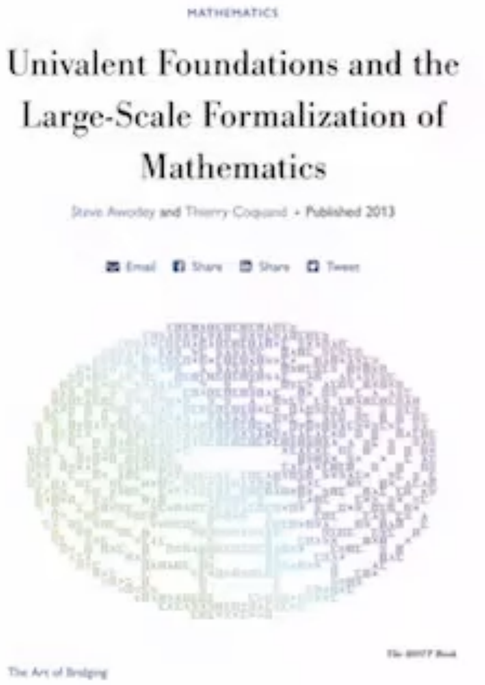
Univariate Polynomials in General Categories

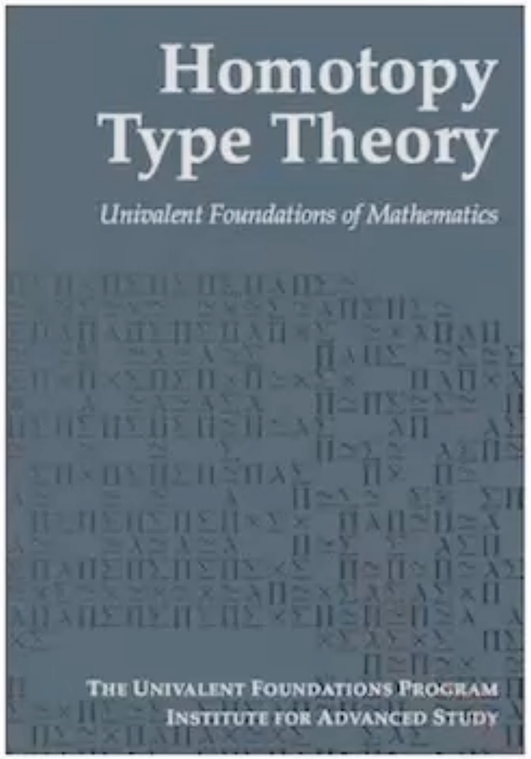
- To compose polynomials we need to prove that the pullback of an exponentiable morphism is exponentiable.
- The only known proof of this is long and uses a technical lemma known as Dubuc's adjoint triangle theorem which provides sufficient conditions for a functor to have a left adjoint, given a triangle of functors.
- We formalized this proof, but during the formalization found a new proof which is about 4 times shorter on pen+paper and about 10 times shorter in LoC.

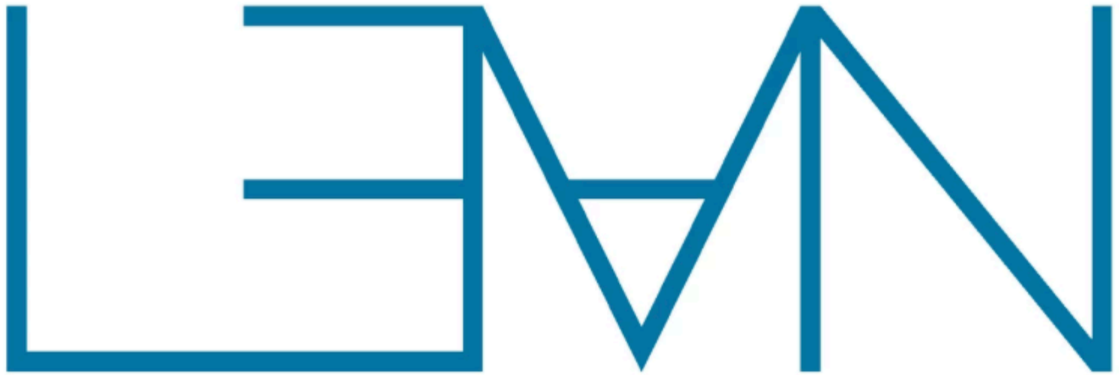
```
def comp {E B D A : C} (P : UvPoly E B) (Q : UvPoly D A) :
    UvPoly (compDom P Q) (P @ A) where
  p := pullback.snd Q.p (fan P A).snd ≫ pullback.fst (fan P A).fst P.p
  exp :=
    haveI := ExponentiableMorphism.of_isPullback (.flip ◁ .of_hasPullback Q.p (fan P A).snd)
    haveI := ExponentiableMorphism.of_isPullback (.of_hasPullback (fan P A).fst P.p)
    inferInstance
```

HoTTLean: Semantics of HoTT in Lean

by S. Awodey, M. Carneiro, S. Hazratpour, J. Hua, W. Nawrocki, S. Rong, S. Woolfson, Y. Xu







An open-source functional programming language and interactive theorem prover.

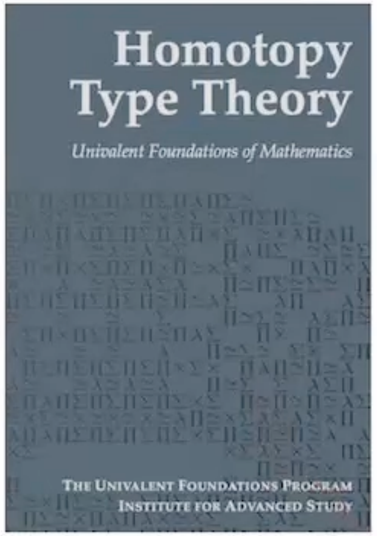
https://sinhp.github.io/HoTTLean

Homotopy Type Theory (HoTT)

HoTT is a foundation of mathematics based on dependent type theory with a new principles of reasoning connecting logic (type theory) and topology (homotopy).

In HoTT, every type encodes the structure of an infinity groupoid.

Space X	→	Type X
Point x in X	→	Term x : X
Path p from x to y in X	→	p : x = y
Homotopy H from p to q	→	H : p = q

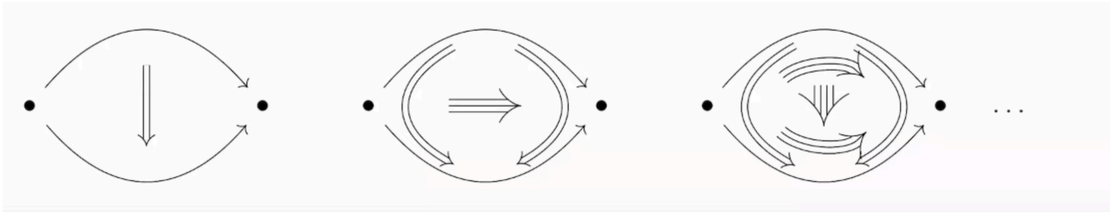


Open collaborative textbook, Special Year on HoTT, IAS 2013

The Fundamental ∞-Groupoid

The fundamental ∞-groupoid of a space X encodes all higher homotopical data of X:

- Objects are points in the space
- Morphisms are continuous paths between points
- 2-morphims are homotopies between paths
- 3-morphisms are homotopies between homotopies, and so on.
- Composition operation is homotopy coherent concatenation of homotopies.



HoTT

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, x{:}A \vdash B : \mathcal{U}_i}{\Gamma \vdash \sum_{(x:A)} B : \mathcal{U}_i}\ \Sigma\text{-FORM}$$

$$\frac{\Gamma, x{:}A \vdash B : \mathcal{U}_i \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : B[a/x]}{\Gamma \vdash (a, b) : \sum_{(x:A)} B}\ \Sigma\text{-INTRO}$$

$$\frac{\Gamma, z{:}\sum_{(x:A)} B \vdash C : \mathcal{U}_i \qquad \Gamma, x{:}A, y{:}B \vdash g : C[(x,y)/z] \qquad \Gamma \vdash p : \sum_{(x:A)} B}{\Gamma \vdash \mathsf{ind}_{\sum_{(x:A)} B}(z.C, x.y.g, p) : C[p/z]}\ \Sigma\text{-ELIM}$$

$$\frac{\begin{array}{c}\Gamma, z{:}\sum_{(x:A)} B \vdash C : \mathcal{U}_i \qquad \Gamma, x{:}A, y{:}B \vdash g : C[(x,y)/z] \\ \Gamma \vdash a : A \qquad \Gamma \vdash b : B[a/x]\end{array}}{\Gamma \vdash \mathsf{ind}_{\sum_{(x:A)} B}(z.C, x.y.g, (a,b)) \equiv g[a,b/x,y] : C[(a,b)/z]}\ \Sigma\text{-COMP}$$

Dependent pair types (Σ-types)

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma, x{:}A \vdash B : \mathcal{U}_i}{\Gamma \vdash \prod_{(x:A)} B : \mathcal{U}_i}\ \Pi\text{-FORM} \qquad \frac{\Gamma, x{:}A \vdash b : B}{\Gamma \vdash \lambda(x:A).b : \prod_{(x:A)} B}\ \Pi\text{-INTRO}$$

$$\frac{\Gamma \vdash f : \prod_{(x:A)} B \qquad \Gamma \vdash a : A}{\Gamma \vdash f(a) : B[a/x]}\ \Pi\text{-ELIM} \qquad \frac{\Gamma, x{:}A \vdash b : B \qquad \Gamma \vdash a : A}{\Gamma \vdash (\lambda(x:A).b)(a) \equiv b[a/x] : B[a/x]}\ \Pi\text{-COMP}$$

$$\frac{\Gamma \vdash f : \prod_{(x:A)} B}{\Gamma \vdash f \equiv (\lambda x.\, f(x)) : \prod_{(x:A)} B}\ \Pi\text{-UNIQ}$$

Dependent function types (Π-types)

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathbf{0} : \mathcal{U}_i}\ \mathbf{0}\text{-FORM} \qquad \frac{\Gamma, x{:}\mathbf{0} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash a : \mathbf{0}}{\Gamma \vdash \mathsf{ind}_{\mathbf{0}}(x.C, a) : C[a/x]}\ \mathbf{0}\text{-ELIM}$$

The empty type

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathbf{1} : \mathcal{U}_i}\ \mathbf{1}\text{-FORM} \qquad \frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \star : \mathbf{1}}\ \mathbf{1}\text{-INTRO} \qquad \frac{\Gamma, x{:}\mathbf{1} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash c : C[\star/x] \qquad \Gamma \vdash a : \mathbf{1}}{\Gamma \vdash \mathsf{ind}_{\mathbf{1}}(x.C, c, a) : C[a/x]}\ \mathbf{1}\text{-ELIM}$$

$$\frac{\Gamma, x{:}\mathbf{1} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash c : C[\star/x]}{\Gamma \vdash \mathsf{ind}_{\mathbf{1}}(x.C, c, \star) \equiv c : C[\star/x]}\ \mathbf{1}\text{-COMP}$$

The unit type

$$\frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash \mathbb{N} : \mathcal{U}_i}\ \mathbb{N}\text{-FORM} \qquad \frac{\Gamma\ \mathsf{ctx}}{\Gamma \vdash 0 : \mathbb{N}}\ \mathbb{N}\text{-INTRO}_1 \qquad \frac{\Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{succ}(n) : \mathbb{N}}\ \mathbb{N}\text{-INTRO}_2$$

$$\frac{\Gamma, x{:}\mathbb{N} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash c_0 : C[0/x] \qquad \Gamma, x{:}\mathbb{N}, y{:}C \vdash c_s : C[\mathsf{succ}(x)/x] \qquad \Gamma \vdash n : \mathbb{N}}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, n) : C[n/x]}\ \mathbb{N}\text{-ELIM}$$

$$\frac{\Gamma, x{:}\mathbb{N} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash c_0 : C[0/x] \qquad \Gamma, x{:}\mathbb{N}, y{:}C \vdash c_s : C[\mathsf{succ}(x)/x]}{\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, 0) \equiv c_0 : C[0/x]}\ \mathbb{N}\text{-COMP}_1$$

$$\frac{\Gamma, x{:}\mathbb{N} \vdash C : \mathcal{U}_i \qquad \Gamma \vdash c_0 : C[0/x] \qquad \Gamma, x{:}\mathbb{N}, y{:}C \vdash c_s : C[\mathsf{succ}(x)/x] \qquad \Gamma \vdash n : \mathbb{N}}{\begin{array}{l}\Gamma \vdash \mathsf{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, \mathsf{succ}(n)) \\ \qquad \equiv c_s[n, \mathsf{ind}_{\mathbb{N}}(x.C, c_0, x.y.c_s, n)/x, y] : C[\mathsf{succ}(n)/x]\end{array}}\ \mathbb{N}\text{-COMP}_2$$

The natural number type

$$\frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A}{\Gamma \vdash a =_A b : \mathcal{U}_i}\ =\text{-FORM} \qquad \frac{\Gamma \vdash A : \mathcal{U}_i \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{refl}_a : a =_A a}\ =\text{-INTRO}$$

$$\frac{\begin{array}{c}\Gamma, x{:}A, y{:}A, p{:}x =_A y \vdash C : \mathcal{U}_i \\ \Gamma, z{:}A \vdash c : C[z, z, \mathsf{refl}_z/x, y, p] \qquad \Gamma \vdash a : A \qquad \Gamma \vdash b : A \qquad \Gamma \vdash p' : a =_A b\end{array}}{\Gamma \vdash \mathsf{ind}_{=_A}(x.y.p.C, z.c, a, b, p') : C[a, b, p'/x, y, p]}\ =\text{-ELIM}$$

$$\frac{\Gamma, x{:}A, y{:}A, p{:}x =_A y \vdash C : \mathcal{U}_i \qquad \Gamma, z{:}A \vdash c : C[z, z, \mathsf{refl}_z/x, y, p] \qquad \Gamma \vdash a : A}{\Gamma \vdash \mathsf{ind}_{=_A}(x.y.p.C, z.c, a, a, \mathsf{refl}_a) \equiv c[a/z] : C[a, a, \mathsf{refl}_a/x, y, p]}\ =\text{-COMP}$$

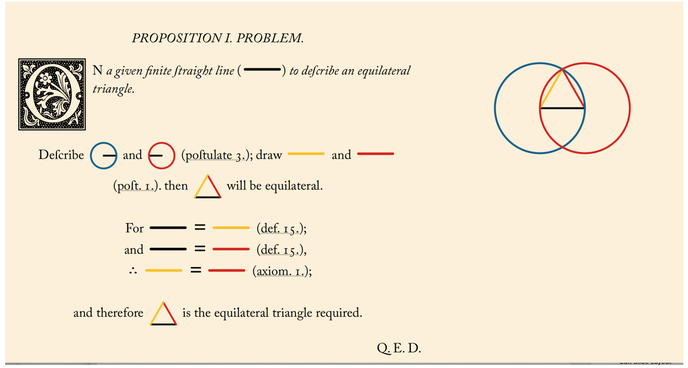
Identity types

The advantage of synthetic methods

- Makes mathematical objects **directly accessible**, instead of carving them out in the specific models.
- We do not get distracted or obstructed by the particularities of the model.
- Synthetic theorems and proofs are **more conceptual** and hold at a higher level of generality. They are often much **shorter** too.

Synthetic vs Analytic Mathematics

Synthetic
(Euclidean Geometry)



PROPOSITION I. PROBLEM.

N *a given finite ſtraight line* (——) *to deſcribe an equilateral triangle.*

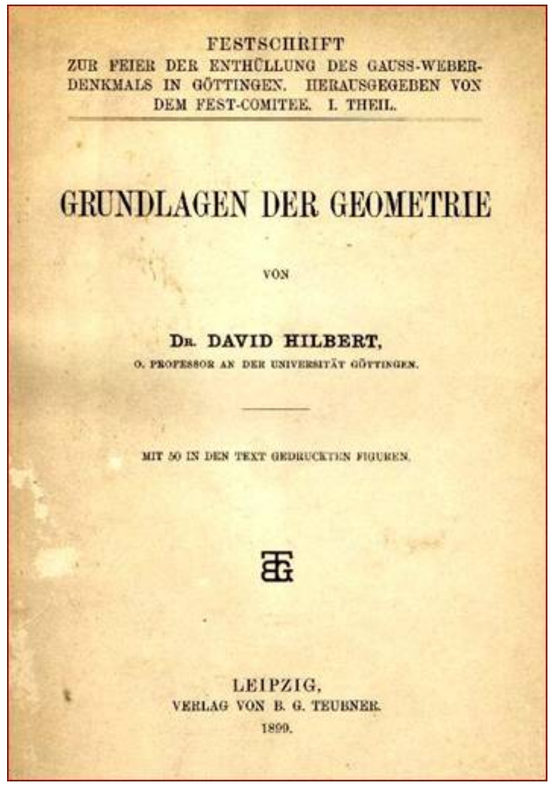
Deſcribe ◯ and ◯ (poſtulate 3.); draw —— and ——
(poſt. 1.). then △ will be equilateral.

For —— = —— (def. 15.);
and —— = —— (def. 15.),
∴ —— = —— (axiom. 1.);

and therefore △ is the equilateral triangle required.

Q. E. D.

From reproduction of Byrne's Euclid by Nicholas Rougeux



FESTSCHRIFT
ZUR FEIER DER ENTHÜLLUNG DES GAUSS-WEBER-DENKMALS IN GÖTTINGEN. HERAUSGEGEBEN VON DEM FEST-COMITEE. I. THEIL.

GRUNDLAGEN DER GEOMETRIE

VON

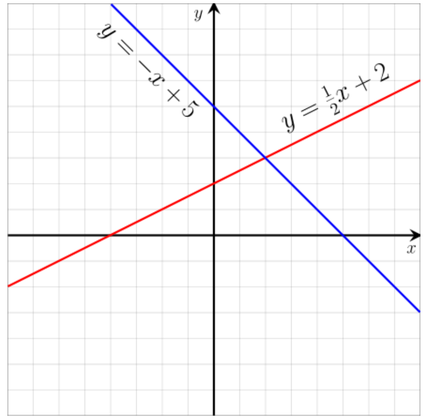
DR. DAVID HILBERT,
O. PROFESSOR AN DER UNIVERSITÄT GÖTTINGEN.

MIT 50 IN DEN TEXT GEDRUCKTEN FIGUREN.

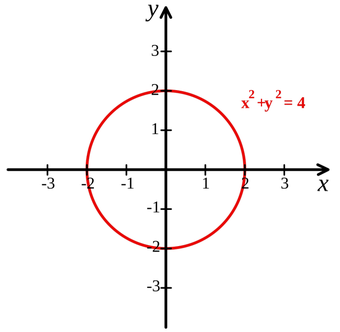
LEIPZIG,
VERLAG VON B. G. TEUBNER.
1899.

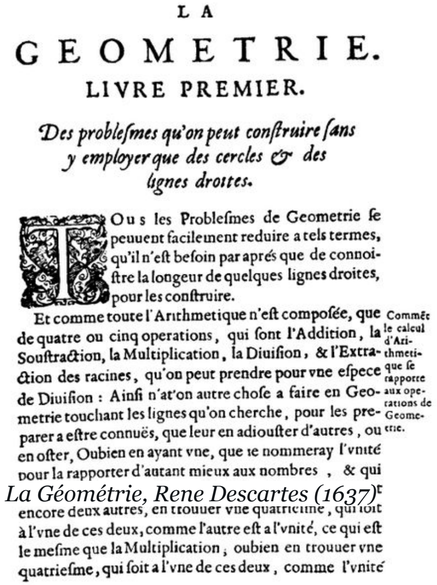
Analytic
(Geometry with coordinates)

$$L := \{(x, y) \mid x + y - 5 = 0\}$$



$$C := \{(x, y) \mid x^2 + y^2 - 4 = 0\}$$





LA
GEOMETRIE.
LIVRE PREMIER.

Des problesmes qu'on peut construire sans y employer que des cercles & des lignes droites.

TOus les Problesmes de Geometrie se peuuent facilement reduire a tels termes, qu'il n'eſt beſoin par aprés que de connoiſtre la longeur de quelques lignes droites, pour les conſtruire.

Et comme toute l'Arithmetique n'eſt compoſée, que de quatre ou cinq operations, qui ſont l'Addition, la Souſtraction, la Multiplication, la Diuiſion, & l'Extraction des racines, qu'on peut prendre pour vne eſpece de Diuiſion : Ainſi n'at'on autre choſe a faire en Geometrie touchant les lignes qu'on cherche, pour les preparer a eſtre connuës, que leur en adiouſter d'autres, ou en oſter, Oubien en ayant vne, que ie nommeray l'vnité pour la rapporter d'autant mieux aux nombres, & qui encore deux autres, en trouuer vne quatrieſme, qui ſoit à l'vne de ces deux, comme l'autre eſt a l'vnité, ce qui eſt le meſme que la Multiplication, oubien en trouuer vne quatrieſme, qui ſoit a l'vne de ces deux, comme l'vnité

Comme le calcul d'Arithmetique ſe rapporte aux operations de Geometrie.

La Géométrie, Rene Descartes (1637)

Synthetic vs Analytic Mathematics

$$P \neq Q : \text{Point}$$

$$\exists (C_1 : \text{Circle}).\ \text{Center}(P, C_1) \wedge Q \in C_1$$

$$\exists (C_2 : \text{Circle}).\ \text{Center}(Q, C_2) \wedge P \in C_2$$

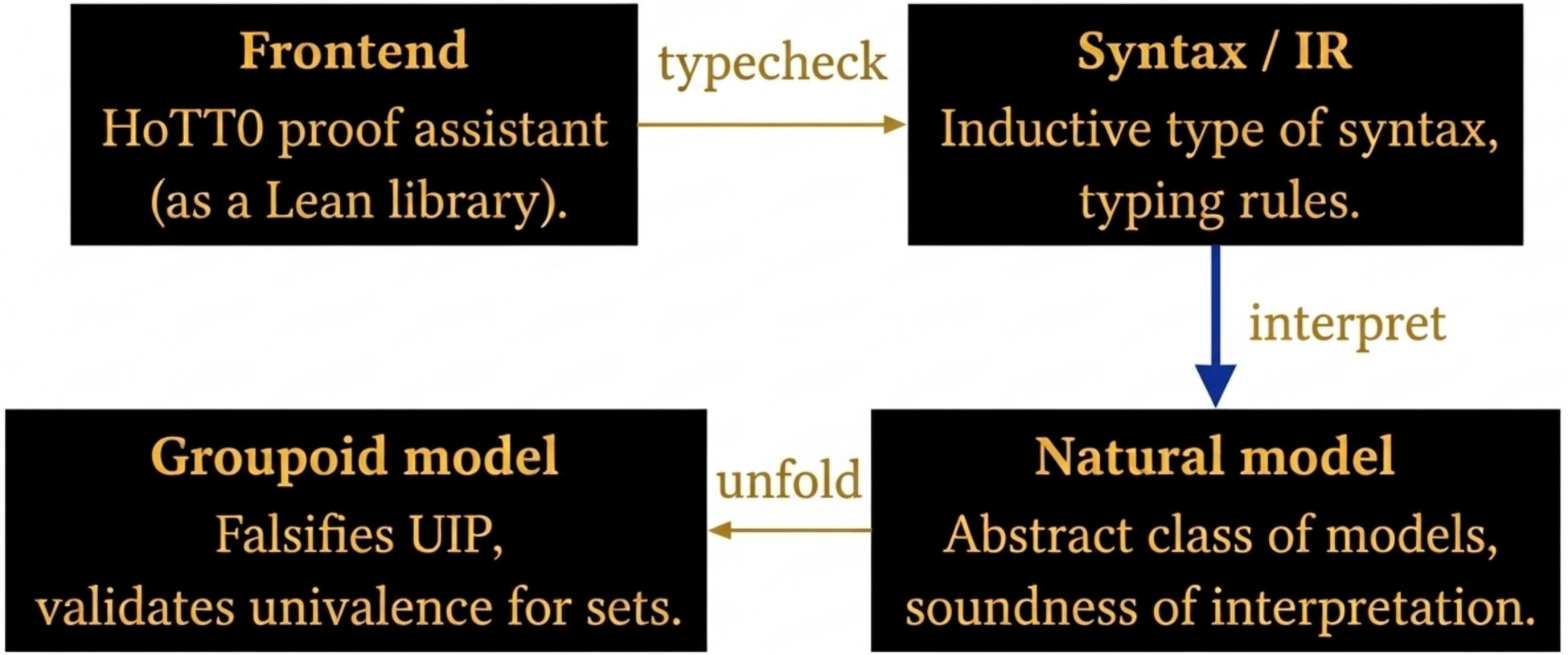
$$\text{Intersect}(C_1, C_2)$$

$$P \neq Q \in \mathbb{R}^2$$

$$C_1 = \left\{(x, y) \mid (x + \ldots)^2 + (y + \ldots)^2 = \ldots\right\}$$

$$C_2 = \left\{(x, y) \mid (x + \ldots)^2 + (y + \ldots)^2 = \ldots\right\}$$

$$\exists R \in C_1 \cap C_2$$


Frontend
HoTT0 proof assistant
(as a Lean library).
typecheck
Syntax / IR
Inductive type of syntax,
typing rules.
interpret
Groupoid model
Falsifies UIP,
validates univalence for sets.
unfold
Natural model
Abstract class of models,
soundness of interpretation.

Syntax and the Front End

```
inductive Expr where
  /-- An axiom (i.e., a closed term constant in the theory) of the given type. -/
  | ax (c : χ) (A : Expr)
  /-- De Bruijn index. -/
  | bvar (i : Nat)
  /-- Dependent product. -/
  | pi (l l' : Nat) (A B : Expr)
  /-- Lambda with the specified argument type. -/
  | lam (l l' : Nat) (A b : Expr)
  /-- Application at the specified output type family `B`. -/
  | app (l l' : Nat) (B fn arg : Expr)
  /-- Dependent sum. -/
  | sigma (l l' : Nat) (A B : Expr)
  /-- Pair formation. -/
  | pair (l l' : Nat) (B t u : Expr)
  /-- First component of a pair. -/
  | fst (l l' : Nat) (A B p : Expr)
  /-- Second component of a pair. -/
  | snd (l l' : Nat) (A B p : Expr)
  /-- Identity type. -/
  -- TODO: capitalize all the types
  | Id (l : Nat) (A t u : Expr)
  /-- A reflexive identity. -/
  | refl (l : Nat) (t : Expr)
  /-- Eliminates an identity. -/
  | idRec (l l' : Nat) (t M r u h : Expr)
  /-- A type universe. -/
  | univ (l : Nat)
  /-- Type from a code. -/
```

```
partial def translate : Lean.Expr → TranslateM Q(HoTT.Expr)

partial def checkEqTm (Γ : Ctx) (t u A : Expr) :
  Except String { _u : Unit // Γ ⊢[0] t ≡ u : A }

elab "hott def" name ":" tp ":=" val : command => do
  let tpE ← elabTerm tp
  let valE ← elabTerm tpE val
  let tpHott ← translate tpE
  let valHott ← translate valE
  checkEqTm [] valHott valHott tpHott
```

Dependent Type Theory (DTT)

Types:

$$A, B, \ \ldots$$

Terms:

$$x : A, \ b : B, \ \ldots$$

Dependent types: ("type-indexed families of types")

$$x : A \vdash B(x)$$

$$x : A, y : B(x) \vdash C(x, y)$$

$$\ldots$$

Type forming operations:

$$\textstyle\sum_{x:A} B(x), \quad \prod_{x:A} B(x), \quad \ldots$$

Term forming operations:

$$\langle a, b \rangle, \quad \lambda x.b(x), \ \ldots$$

Equations:

$$s = t : A$$

Contexts:

$$\frac{x : A \vdash B(x)}{x : A, \ y : B(x) \vdash}$$

Writing Γ for any context, we have:

$$\frac{\Gamma \vdash C}{\Gamma, z : C \vdash}$$

DTT: Rules

Sums:

$$\frac{x : A \vdash B(x)}{\sum_{x:A} B(x)} \qquad \frac{a : A \qquad b : B(a)}{\langle a, b \rangle : \sum_{x:A} B(x)}$$

$$\frac{c : \sum_{x:A} B(x)}{\mathsf{fst}\, c : A} \qquad \frac{c : \sum_{x:A} B(x)}{\mathsf{snd}\, c : B(\mathsf{fst}\, c)}$$

$$\mathsf{fst}\langle a, b \rangle = a : A \qquad \mathsf{snd}\langle a, b \rangle = b : B$$

$$\langle \mathsf{fst}\, c, \mathsf{snd}\, c \rangle = c : \sum_{x:A} B(x)$$

DTT: Rules

Products:

$$\frac{x : A \vdash B(x)}{\prod_{x:A} B(x)} \qquad \frac{x : A \vdash b(x) : B(x)}{\lambda x.b(x) : \prod_{x:A} B(x)}$$

$$\frac{a : A \qquad f : \prod_{x:A} B(x)}{fa : B(a)}$$

$$x : A \vdash (\lambda x.b)x = b : B(x)$$

$$\lambda x.fx = f : \prod_{x:A} B(x)$$

DTT: Substitution

A tuple of terms in context $\sigma : \Delta \to \Gamma$ induces an operation

$$\frac{\sigma : \Delta \to \Gamma \qquad \Gamma \vdash a : A}{\Delta \vdash a[\sigma] : A[\sigma]}$$

which preserves *everything*.

For example given $y : Y \vdash s : Z$ and $z : Z, x : A(z) \vdash B(z, x)$ we can do

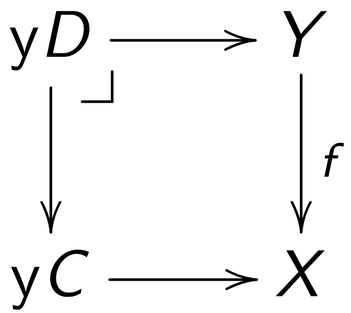
$$\frac{y : Y \vdash s : Z \quad \frac{z{:}Z, x{:}A(z) \vdash B(z,x)}{z{:}Z \vdash \prod_{x:A(z)} B(z,x)}}{y : Y \vdash (\prod_{x:A(z)} B(z, x))[s/z]} \quad \text{or} \quad \frac{\frac{y{:}Y \vdash s{:}Z \quad z{:}Z, x{:}A(z) \vdash B(z,x)}{y{:}Y, x{:}A(s) \vdash B(s,x)}}{y : Y \vdash \prod_{x:A(s)} B(s, x)}$$

and syntactically the results are *the same*,

$$\big(\prod_{x:A(z)} B(z, x)\big)[s/z] = \prod_{x:A(s)} B(s, x)\,.$$

Representable Natural Transformation

A natural transformation $f : Y \to X$ of presheaves on a category $\mathbb{C}$ is called *representable* if its pullback along any $\mathrm{y}C \to X$ is representable:



Natural Model (Awodey, 2014)

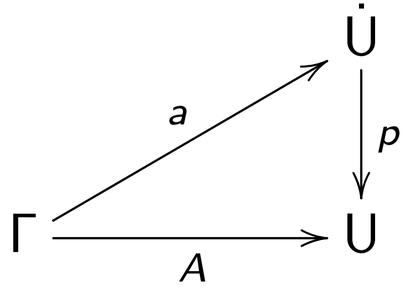
Write the objects and arrows of $\mathbb{C}$ as $\sigma : \Delta \to \Gamma$, thinking of a *category of contexts and substitutions*.

Let $p : \dot{\mathsf{U}} \to \mathsf{U}$ be a representable map of presheaves on $\mathbb{C}$.

Think of U as the *presheaf of types*, $\dot{\mathsf{U}}$ as the *presheaf of terms*, and then p gives the type of a term.

$$\begin{aligned} \Gamma \vdash A \quad &\approx \quad A \in \mathsf{U}(\Gamma) \\ \Gamma \vdash a : A \quad &\approx \quad a \in \dot{\mathsf{U}}(\Gamma) \end{aligned}$$

where $A = p \circ a$.



Natural Model: Context Extension

The remaining operation of **context extension**

$$\frac{\Gamma \vdash A}{\Gamma, x : A \vdash}$$

is modeled by the representability of $p : \dot{\mathsf{U}} \to \mathsf{U}$ as follows.

Let $p_A : \Gamma.A \to \Gamma$ be the pullback of p along A.

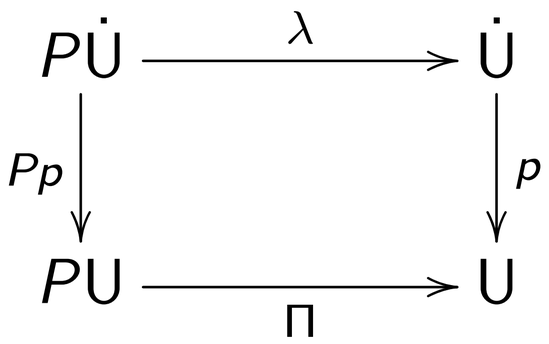
$$\begin{array}{ccc} \Gamma.A & \xrightarrow{q_A} & \dot{\mathsf{U}} \\ {\scriptstyle p_A}\downarrow & \lrcorner & \downarrow{\scriptstyle p} \\ \Gamma & \xrightarrow[A]{} & \mathsf{U} \end{array}$$

The map $q_A : \Gamma.A \to \dot{\mathsf{U}}$ gives the required term $\Gamma.A \vdash q_A : A[p_A]$.
Syntactically, this is just the term

$$\Gamma, x : A \vdash x : A\,.$$

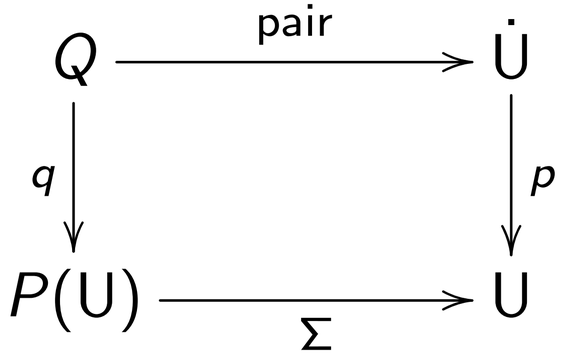
Type Formers via Natural Model

The natural model $p : \dot{\mathsf{U}} \to \mathsf{U}$ *models the rules for products just if there are maps* λ, Π *making the following a pullback.*



Type Formers via Natural Model

The map $p : \dot{\mathsf{U}} \to \mathsf{U}$ *models the rules for sums just if there are maps* (pair, Σ) *making the following a pullback*



where $q = p \triangleleft p : Q \to P(\mathsf{U})$ *is the generating map of the composite* $P_q = P_{p \triangleleft p} = P_p \circ P_p$.

Natural Model in Lean4

```
variable (Ctx) in
class NaturalModelBase where
  Tm : Psh Ctx
  Ty : Psh Ctx
  tp : Tm → Ty
  ext (Γ : Ctx) (A : y(Γ) → Ty) : Ctx
  disp (Γ : Ctx) (A : y(Γ) → Ty) : ext Γ A → Γ
  var (Γ : Ctx) (A : y(Γ) → Ty) : y(ext Γ A) → Tm
  disp_pullback {Γ : Ctx} (A : y(Γ) → Ty) :
    IsPullback (var Γ A) (yoneda.map (disp Γ A)) tp A
```

Natural Model with (Sigma, Pi, Id)-Type Formers

```
variable (Ctx) in      You, 19 months ago • moving poly here
class NaturalModelPi where
  Pi : (P tp).obj Ty → M.Ty
  lam : (P tp).obj Tm → M.Tm
  Pi_pullback : IsPullback lam ((P tp).map tp) tp Pi

variable (Ctx) in
class NaturalModelSigma where
  Sig : (P tp).obj Ty → M.Ty
  pair : (P tp).obj Tm → M.Tm
  Sig_pullback : IsPullback pair ((uvPoly tp).comp (uvPoly tp)).p tp Sig

def δ : M.Tm → pullback tp tp := pullback.lift (𝟙 _) (𝟙 _) rfl
variable (Ctx) in
class NaturalModelEq where
  Eq : pullback tp tp → M.Ty
  refl : Tm → M.Tm
  Eq_pullback : IsPullback refl δ tp Eq

variable (Ctx) in
class NaturalModelIdBase where
  Id : pullback tp tp → M.Ty
  i : Tm → M.Tm
  Id_commute : δ ≫ Id = i ≫ tp
```

```
variable (Ctx) in
class NaturalModel extends
  NaturalModelBase Ctx, NaturalModelPi Ctx, NaturalModelSigma Ctx,
  NaturalModelId Ctx, NaturalModelU Ctx, NaturalModelSmallPi Ctx
```

What to Formalize Next

- Multivariable Polynomials: definition, basic API and composition of multivariable polynomials are done. The best categorical setting is double categories, which needs to be done.
- Induction, Well-founded Trees, Transfinite Induction
- Polynomial functors and the Semantics of Linear Logic
- Polynomial functors and Generalized Combinatorial Species
- Formal Differentiation and The Calculus of Data Structures (Zippers), Semantics of Memory Management
- Applications in Dynamical Systems, Open Games, Lenses

References

- HoTTLean: Formalizing the Meta-Theory of HoTT in Lean 4
  Types 2025
  with S.Awodey, M.Carneiro, W.Nawrocki, S.Woolfson, and Y.Xu

- A Certifying Proof Assistant for Synthetic Mathematics in Lean
  CPP 2026
  with S. Awodey, M. Carneiro, J. Hua, W. Nawrocki, Shuge Rong , Yiming Xu

- Polynomial Functors in Lean 4
  https://github.com/sinhp/Poly

Thank you for your attention!