Type Theory as Foundations of Proof Assistants

(Lean seminar , BI)

Dan Frumin

Bernoulli Institute, University of Groningen

23 Oct, 2026

Formal / Mechanized Proofs

- Used to be important for computer science and logicians
- But now mathematicians are getting on board!

Formal / Mechanized Proofs

- Used to be important for computer science and logicians

- But now mathematicians are getting on board!
  - Proofs in mathematics are conceptual building on large body of previous results, usually not a lot of computations or case analysis
  - The statements and targets are “immutable”

Formal / Mechanized Proofs

- Used to be important for computer science and logicians
  - Many proofs in CS involve large case analyses, but often standalone
  - The targets are moving!

- But now mathematicians are getting on board!
  - Proofs in mathematics are conceptual building on large body of previous results, usually not a lot of computations or case analysis
  - The statements and targets are “immutable”

ZFC vs Type Theory

ZFC:

- First-order logic as the base
- Set-theoretic symbols and axioms of top
- Preferred treatment: few rules, many axioms
- Far away from “informal” mathematics

ZFC vs Type Theory

ZFC:

- First-order logic as the base
- Set-theoretic symbols and axioms of top
- Preferred treatment: few rules, many axioms
- Far away from “informal” mathematics

- Type Theory is what mathematics can reduced to in practice.
- Type Theory combines both proofs and computations
- Proof checking reduces to known algorithmic problems

What I present in this talk is minimal/basic Martin-Löf Type Theory, the type theory of Lean is a bit different.

ZFC vs Type Theory

- For the purposes of this talk we will intuitively think of types = set
- Write $e : A$ instead of $e \in A$

ZFC vs Type Theory

- For the purposes of this talk we will intuitively think of types = set
- Write $e : A$ instead of $e \in A$
- Constructions in set theory: $\emptyset$, $\{a, b\}$, $\cup$, $\cap$, $\mathcal{P}(A)$
- Constructions in type theory: different collection of constructions

ZFC vs Type Theory

- For the purposes of this talk we will intuitively think of types = set
- Write $e : A$ instead of $e \in A$
- Constructions in set theory: $\emptyset$, $\{a, b\}$, $\cup$, $\cap$, $\mathcal{P}(A)$
- Constructions in type theory: different collection of constructions
- Constructions in set theory are about what kinds of sets exist

$$\exists I\big(\emptyset \in I \wedge \forall x(x \in I \to x \cup \{x\} \in I)\big)$$

ZFC vs Type Theory

- For the purposes of this talk we will intuitively think of types = set
- Write $e : A$ instead of $e \in A$
- Constructions in set theory: $\emptyset$, $\{a, b\}$, $\cup$, $\cap$, $\mathcal{P}(A)$
- Constructions in type theory: different collection of constructions
- Constructions in set theory are about what kinds of sets exist

$$\exists I\big(\emptyset \in I \wedge \forall x(x \in I \to x \cup \{x\} \in I)\big)$$

- Constructions in type theory are given directly through operations:

  We have a type $\mathbb{N}$ + we have ways of constructing elements of $\mathbb{N}$
  This gives type theory a more “algebraic” or “categorical” feel

Judgments in Type Theory

Γ ctx	Γ is a well-formed context
$\Gamma \vdash A$ type	A is a type in context Γ
$\Gamma \vdash a : A$	a is a term of type A
$\Gamma \vdash A \equiv B$	A and B are equal types
$\Gamma \vdash a \equiv b : A$	a and b are equal terms of type A

Judgments in Type Theory

Γ ctx	Γ is a well-formed context
$\Gamma \vdash A$ type	A is a type in context Γ
$\Gamma \vdash a : A$	a is a term of type A
$\Gamma \vdash A \equiv B$	A and B are equal types
$\Gamma \vdash a \equiv b : A$	a and b are equal terms of type A

A context records the assumptions currently available. We will first see these judgments in a setting where no assumptions are needed.

Warm up: Cartesian product

$$\frac{A \text{ type} \qquad B \text{ type}}{A \times B \text{ type}}$$

Warm up: Cartesian product

$$\frac{A \text{ type} \qquad B \text{ type}}{A \times B \text{ type}} \qquad \frac{a : A \qquad b : B}{(a, b) : A \times B}$$

Warm up: Cartesian product

$$\frac{A \text{ type} \qquad B \text{ type}}{A \times B \text{ type}} \qquad \frac{a : A \qquad b : B}{(a, b) : A \times B} \qquad \frac{u : A \times B}{\pi_1(u) : A} \qquad \frac{u : A \times B}{\pi_2(u) : B}$$

Warm up: Cartesian product

$$\frac{A \text{ type} \qquad B \text{ type}}{A \times B \text{ type}} \qquad \frac{a : A \qquad b : B}{(a, b) : A \times B} \qquad \frac{u : A \times B}{\pi_1(u) : A} \qquad \frac{u : A \times B}{\pi_2(u) : B}$$

$$\frac{a : A \qquad b : B}{\pi_1((a, b)) \equiv a : A} \qquad \frac{a : A \qquad b : B}{\pi_2((a, b)) \equiv b : B} \qquad \frac{u : A \times B}{(\pi_1(u), \pi_2(u)) \equiv u : A \times B}$$

Warm up: Sum types/disjoint unions

$$\frac{A \text{ type} \qquad B \text{ type}}{A + B \text{ type}}$$

Warm up: Sum types/disjoint unions

$$\frac{A \text{ type} \qquad B \text{ type}}{A + B \text{ type}} \qquad \frac{a : A}{\iota_1(a) : A + B} \qquad \frac{b : B}{\iota_2(b) : A + B}$$

Warm up: Sum types/disjoint unions

$$\frac{A \text{ type} \qquad B \text{ type}}{A + B \text{ type}} \qquad \frac{a : A}{\iota_1(a) : A + B} \qquad \frac{b : B}{\iota_2(b) : A + B}$$

$$\frac{u : A + B \qquad x : A \vdash e_1 : C \qquad y : B \vdash e_2 : C}{\text{case } u \text{ of } \iota_1(x) \Rightarrow e_1 \mid \iota_2(y) \Rightarrow e_2 : C}$$

Warm up: Sum types/disjoint unions

$$\frac{A \text{ type} \qquad B \text{ type}}{A + B \text{ type}} \qquad \frac{a : A}{\iota_1(a) : A + B} \qquad \frac{b : B}{\iota_2(b) : A + B}$$

$$\frac{u : A + B \qquad x : A \vdash e_1 : C \qquad y : B \vdash e_2 : C}{\text{case } u \text{ of } \iota_1(x) \Rightarrow e_1 \mid \iota_2(y) \Rightarrow e_2 : C}$$

$$\left(\text{case } \iota_1(a) \text{ of } \iota_1(x) \Rightarrow e_1 \mid \iota_2(y) \Rightarrow e_2\right) \equiv e_1[a/x] : C$$

Warm up: Sum types/disjoint unions

$$\frac{A \text{ type} \qquad B \text{ type}}{A + B \text{ type}} \qquad \frac{a : A}{\iota_1(a) : A + B} \qquad \frac{b : B}{\iota_2(b) : A + B}$$

$$\frac{u : A + B \qquad x : A \vdash e_1 : C \qquad y : B \vdash e_2 : C}{\text{case } u \text{ of } \iota_1(x) \Rightarrow e_1 \mid \iota_2(y) \Rightarrow e_2 : C}$$

$$\big(\text{case } \iota_1(a) \text{ of } \iota_1(x) \Rightarrow e_1 \mid \iota_2(y) \Rightarrow e_2\big) \equiv e_1[a/x] : C$$

$$\big(\text{case } u \text{ of } \iota_1(x) \Rightarrow \iota_1(x) \mid \iota_2(y) \Rightarrow \iota_2(y)\big) \equiv u : A + B$$

Natural numbers

$\mathbb{N}$ type

Natural numbers

$\mathbb{N}$ type $\qquad 0 : \mathbb{N} \qquad \dfrac{n : \mathbb{N}}{S(n) : \mathbb{N}}$

Natural numbers

$$\mathbb{N}\ \text{type} \qquad 0 : \mathbb{N} \qquad \frac{n : \mathbb{N}}{S(n) : \mathbb{N}} \qquad \frac{n : \mathbb{N} \qquad e_0 : C \qquad x : C \vdash e_S : C}{\mathrm{rec}_{\mathbb{N}}(e_0, e_S, n) : C}$$

Natural numbers

$$\mathbb{N}\ \text{type} \qquad 0 : \mathbb{N} \qquad \frac{n : \mathbb{N}}{S(n) : \mathbb{N}} \qquad \frac{n : \mathbb{N} \qquad e_0 : C \qquad x : C \vdash e_S : C}{\mathrm{rec}_{\mathbb{N}}(e_0, e_S, n) : C}$$

$$\mathrm{rec}_{\mathbb{N}}(e_0, e_S, 0) \equiv e_0 : C \qquad \mathrm{rec}_{\mathbb{N}}(e_0, e_S, S(n)) \equiv e_S(\mathrm{rec}_{\mathbb{N}}(e_0, e_S, n)) : C$$

$$\mathrm{rec}_{\mathbb{N}}(0, S, n) \equiv n : \mathbb{N}$$

Addition by recursion

We can define addition recursively as

$$0 + m = m$$
$$S(n) + m = S(n + m)$$

Addition by recursion

We can define addition recursively as

$$0 + m = m$$
$$S(n) + m = S(n + m)$$

$$\frac{m : \mathbb{N} \qquad x : \mathbb{N} \vdash S(x) : \mathbb{N}}{\mathrm{rec}_{\mathbb{N}}(m, S, n) : \mathbb{N}}$$

Function types

$$\frac{A \text{ type} \qquad B \text{ type}}{A \to B \text{ type}}$$

Function types

$$\frac{A \text{ type} \qquad B \text{ type}}{A \to B \text{ type}} \qquad \frac{x : A \vdash e : B}{\lambda x.e : A \to B}$$

Function types

$$\frac{A \text{ type} \qquad B \text{ type}}{A \to B \text{ type}} \qquad \frac{x : A \vdash e : B}{\lambda x.e : A \to B} \qquad \frac{t : A \to B \qquad a : A}{t(a) : B}$$

Function types

$$\frac{A \text{ type} \qquad B \text{ type}}{A \to B \text{ type}} \qquad \frac{x : A \vdash e : B}{\lambda x.e : A \to B} \qquad \frac{t : A \to B \qquad a : A}{t(a) : B}$$

$$(\lambda x.e)(a) \equiv e[a/x] : B$$

Contexts and dependencies

We have seen the use of hypothetical judgments $x : A \vdash e : B$, or $x : A \vdash e(x) : B$.
Another kind of dependency: type families

- In mathematics we sometimes works with set-families $A : I \to Set$
- "Let $m \in \mathbb{N}$, and consider a vector $v \in \mathbb{R}^n$.."
- "Consider a point x in a fiber $E(b)$ over a base point b..."

Contexts and dependencies

We have seen the use of hypothetical judgments $x : A \vdash e : B$, or $x : A \vdash e(x) : B$.
Another kind of dependency: type families

- In mathematics we sometimes works with set-families $A : I \to Set$
- "Let $m \in \mathbb{N}$, and consider a vector $v \in \mathbb{R}^n$.."
- "Consider a point x in a fiber $E(b)$ over a base point b..."
- Hypothetical type judgments $x : I \vdash A(i)$ type

Contexts and dependencies

We have seen the use of hypothetical judgments $x : A \vdash e : B$, or $x : A \vdash e(x) : B$.
Another kind of dependency: type families

- In mathematics we sometimes works with set-families $A : I \to Set$
- "Let $m \in \mathbb{N}$, and consider a vector $v \in \mathbb{R}^n$.."
- "Consider a point x in a fiber $E(b)$ over a base point b..."
- Hypothetical type judgments $x : I \vdash A(i)$ type

$$\frac{\Gamma \text{ ctx} \qquad \Gamma \vdash A \text{ type}}{\Gamma, x : A \text{ ctx}} \qquad \frac{\Gamma \text{ ctx} \qquad x : A \in \Gamma}{\Gamma \vdash x : A}.$$

Dependent sum

$$\frac{A \text{ type} \qquad x : A \vdash B(x) \text{ type}}{\sum_{x:A} B(x) \text{ type}}$$

Dependent sum

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

Dependent sum

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

$$\frac{u : \sum_{x:A} B(x)}{\pi_1(u) : A} \qquad \frac{u : \sum_{x:A} B(x)}{\pi_2(u) : B(\pi_1(u))}$$

Dependent sum

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

$$\frac{u : \sum_{x:A} B(x)}{\pi_1(u) : A} \qquad \frac{u : \sum_{x:A} B(x)}{\pi_2(u) : B(\pi_1(u))}$$

Why is it called a sum?

Dependent sum

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

$$\frac{u : \sum_{x:A} B(x)}{\pi_1(u) : A} \qquad \frac{u : \sum_{x:A} B(x)}{\pi_2(u) : B(\pi_1(u))}$$

Why is it called a sum?

$$\sum_{i:\{1,2\}} A_i \simeq A_1 + A_2$$

$$\sum_{i:I} A \simeq I \times A$$

Dependent product

$$\frac{A \text{ type} \qquad x : A \vdash B(x) \text{ type}}{\prod_{x:A} B(x) \text{ type}}$$

Dependent product

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

Dependent product

$$\frac{A \text{ type} \qquad x : A \vdash B(x) \text{ type}}{\prod_{x:A} B(x) \text{ type}}$$

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

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

Dependent product

$$\frac{A \text{ type} \qquad x : A \vdash B(x) \text{ type}}{\prod_{x:A} B(x) \text{ type}} \qquad \frac{x : A \vdash e : B(x)}{\lambda x.e : \prod_{x:A} B(x)} \qquad \frac{f : \prod_{x:A} B(x) \qquad a : A}{f(a) : B(a)}$$

$$\frac{x : A \vdash e : B(x) \qquad a : A}{(\lambda x.e)(a) \equiv e[a/x] : B(a)}$$

Dependent product

$$\frac{A \text{ type} \qquad x : A \vdash B(x) \text{ type}}{\prod_{x:A} B(x) \text{ type}} \qquad \frac{x : A \vdash e : B(x)}{\lambda x.e : \prod_{x:A} B(x)} \qquad \frac{f : \prod_{x:A} B(x) \qquad a : A}{f(a) : B(a)}$$

$$\frac{x : A \vdash e : B(x) \qquad a : A}{(\lambda x.e)(a) \equiv e[a/x] : B(a)}$$

Why is it called a product?

Example: the type of monoids

$$\mathsf{Monoid} := \sum_{G:\mathsf{Type}} \mathsf{MonoidStr}(G)$$

A monoid structure on G:

$$\mathsf{MonoidStr}(G) := \sum_{\mu:G\times G\to G} \sum_{e:G} \mathsf{MonoidLaws}(G,\mu,e)$$

Monoid laws:

$$\mathsf{MonoidLaws}(G,\mu,e) = \mathrm{Associativity}(G,\mu) \times \mathrm{Identity}(G,\mu,e)$$

Where is the logic?

But, Dan, why are we doing programming? I thought we were going to do logic?

Where is the logic?

But, Dan, why are we doing programming? I thought we were going to do logic?

Logic emerges as a natural class of types when

- If A and B are propositions, then $A \wedge B$ is a proposition
- To prove $A \wedge B$ we have to prove A and prove B

Where is the logic?

But, Dan, why are we doing programming? I thought we were going to do logic?

Logic emerges as a natural class of types when

- If A and B are propositions, then $A \wedge B$ is a proposition
- To prove $A \wedge B$ we have to prove A and prove B
- If A and B are types, then $A \times B$ is a type
- To provide an element of $A \times B$ we provide an element of A and an element of B

Where is the logic?

But, Dan, why are we doing programming? I thought we were going to do logic?

Logic emerges as a natural class of types when

- If A and B are propositions, then $A \wedge B$ is a proposition
- To prove $A \wedge B$ we have to prove A and prove B
- If A and B are types, then $A \times B$ is a type
- To provide an element of $A \times B$ we provide an element of A and an element of B
- To prove $A \vee B$ we need to either prove A or prove B
- To provide an element of $A + B$ we provide an element of A or an element of B

Propositions as types

Propositions	Types
$A \wedge B$	product type $A \times B$
$A \vee B$	sum type $A + B$
$A \to B$	function type $A \to B$
$\top$ (always true)	unit type 1
$\bot$ (false)	empty type Empty
$\forall x P(x)$	
$\exists x P(x)$	

Propositions as types

Propositions	Types
$A \wedge B$	product type $A \times B$
$A \vee B$	sum type $A + B$
$A \to B$	function type $A \to B$
$\top$ (always true)	unit type 1
$\bot$ (false)	empty type Empty
$\forall x P(x)$	dep product $\prod_{x:A} P(x)$
$\exists x P(x)$	dep product $\sum_{x:A} P(x)$

Propositions as types

Propositions	Types
$A \wedge B$	product type $A \times B$
$A \vee B$	sum type $A + B$
$A \to B$	function type $A \to B$
$\top$ (always true)	unit type 1
$\bot$ (false)	empty type Empty
$\forall x P(x)$	dep product $\prod_{x:A} P(x)$
$\exists x P(x)$	dep product $\sum_{x:A} P(x)$

In Lean the stratification is more explicit, there are different *universes*: Prop vs Type, where Prop contains *subsingleton* types

Monoid laws

Now we can complete our definition of monoids:

$$\mathsf{Monoid} := \sum_{G:\mathsf{Type}} \sum_{\mu:G\times G\to G} \sum_{e:G} \mathsf{MonoidLaws}(G,\mu,e)$$

$$\mathsf{MonoidLaws}(G,\mu,e) \equiv \begin{pmatrix} (\prod_{x,y,z:G} \mu(\mu(x,y),z) = \mu(x,\mu(y,z))) \times \\ (\prod_{x:G} \mu(e,x) = x) \times \\ (\prod_{x:G} \mu(x,e) = x) \end{pmatrix}$$

Natural numbers, revisited

$$\mathbb{N}\ \text{type} \qquad 0 : \mathbb{N} \qquad \frac{n : \mathbb{N}}{S(n) : \mathbb{N}} \qquad \frac{n : \mathbb{N} \qquad C\ \text{type} \qquad e_0 : C \qquad x : C \vdash e_S : C}{\text{rec}_{\mathbb{N}}(e_0, e_S, n) : C}$$

Natural numbers, revisited

$$\mathbb{N}\ \text{type} \qquad 0 : \mathbb{N} \qquad \frac{n : \mathbb{N}}{S(n) : \mathbb{N}} \qquad \frac{n : \mathbb{N} \qquad C\ \text{type} \qquad e_0 : C \qquad x : C \vdash e_S : C}{\text{rec}_{\mathbb{N}}(e_0, e_S, n) : C}$$

$$\frac{n : \mathbb{N} \qquad x : \mathbb{N} \vdash C(x)\ \text{type} \qquad e_0 : C(0) \qquad n : \mathbb{N}, x : C(n) \vdash e_S : C(S(n))}{\text{rec}_{\mathbb{N}}(e_0, e_S, n) : C(n)}$$

Sums, revisited

$$\frac{A \text{ type} \qquad B \text{ type}}{A + B \text{ type}}$$

Sums, revisited

$$\frac{A \text{ type} \qquad B \text{ type}}{A + B \text{ type}} \qquad \frac{a : A}{\iota_1(a) : A + B} \qquad \frac{b : B}{\iota_2(b) : A + B}$$

Sums, revisited

$$\frac{A \text{ type} \qquad B \text{ type}}{A + B \text{ type}} \qquad \frac{a : A}{\iota_1(a) : A + B} \qquad \frac{b : B}{\iota_2(b) : A + B}$$

$$\frac{u : A + B \qquad C \text{ type} \qquad x : A \vdash e_1 : C \qquad y : B \vdash e_2 : C}{\mathrm{rec}_{A+B}(\lambda x.e_1, \lambda y.e_2, u) : C}$$

Sums, revisited

$$\frac{A \text{ type} \qquad B \text{ type}}{A + B \text{ type}} \qquad \frac{a : A}{\iota_1(a) : A + B} \qquad \frac{b : B}{\iota_2(b) : A + B}$$

$$\frac{u : A + B \qquad C \text{ type} \qquad x : A \vdash e_1 : C \qquad y : B \vdash e_2 : C}{\mathrm{rec}_{A+B}(\lambda x.e_1, \lambda y.e_2, u) : C}$$

$$\frac{\begin{array}{c} u : A + B \qquad w : A + B \vdash C(w) \text{ type} \\ x : A \vdash e_1 : C(\iota_1(x)) \qquad y : B \vdash e_2 : C(\iota_2(y)) \end{array}}{\mathrm{rec}_{A+B}(\lambda x.e_1, \lambda y.e_2, u) : C(u)}$$

Products, revisited

$$\frac{A \text{ type} \qquad B \text{ type}}{A \times B \text{ type}} \qquad\qquad \frac{a : A \qquad b : B}{(a, b) : A \times B}$$

$$\frac{u : A \times B \qquad x : A \times B \vdash C(x) \qquad x : A, y : B \vdash t : C((x, y))}{\mathrm{rec}_{A \times B}(\lambda x \lambda y.t, u) : C(u)}$$

Products, revisited

$$\frac{A \text{ type} \qquad B \text{ type}}{A \times B \text{ type}} \qquad\qquad \frac{a : A \qquad b : B}{(a, b) : A \times B}$$

$$\frac{u : A \times B \qquad x : A \times B \vdash C(x) \qquad x : A, y : B \vdash t : C((x, y))}{\mathrm{rec}_{A \times B}(\lambda x \lambda y.t, u) : C(u)}$$

Projections redefined:

$$\pi_1(u) \equiv \mathrm{rec}_{A \times B}(\lambda x \lambda y.x, u) : A$$

$$\pi_2(u) \equiv \mathrm{rec}_{A \times B}(\lambda x \lambda y.y, u) : B$$

General Inductive Type Format

$$x : A \vdash T(x) \text{ type} \qquad \vec{x} : \vec{B} \vdash c_1(\vec{x}) : T(f_1(\vec{x})) \qquad \cdots \qquad \vec{x} : \vec{B} \vdash c_n(\vec{x}) : T(f_n(\vec{x}))$$

General Inductive Type Format

$$x : A \vdash T(x)\ \text{type} \qquad \vec{x} : \vec{B} \vdash c_1(\vec{x}) : T(f_1(\vec{x})) \qquad \cdots \qquad \vec{x} : \vec{B} \vdash c_n(\vec{x}) : T(f_n(\vec{x}))$$

$$\frac{\begin{array}{c} a : A \qquad u : T(a) \qquad x : A, y : T(a) \vdash C(x, y)\ \text{type} \\ \vec{x} : \vec{B} \vdash e_1 : C(f_1(\vec{x}), c_1(\vec{x})) \qquad \cdots \qquad \vec{x} : \vec{B} \vdash e_n : C(f_n(\vec{x}), c_n(\vec{x})) \end{array}}{\mathrm{rec}_T(\lambda\vec{x}.e_1, \ldots, \lambda\vec{x}.e_n, u) : C(a, u)}$$

+ equations for rec

The identity type

$$\frac{A \text{ type} \qquad a : A \qquad b : A}{\mathrm{Id}_A(a, b) \text{ type}}$$

The identity type

$$\frac{A \text{ type} \qquad a : A \qquad b : A}{\mathrm{Id}_A(a, b) \text{ type}}$$

$$\frac{a : A}{\mathrm{refl}_a : \mathrm{Id}_A(a, a)}$$

The identity type

$$\frac{A\ \text{type} \qquad a : A \qquad b : A}{\mathrm{Id}_A(a, b)\ \text{type}} \qquad\qquad \frac{a : A}{\mathrm{refl}_a : \mathrm{Id}_A(a, a)}$$

$$\frac{a : A \qquad b : A \qquad p : \mathrm{Id}_A(a, b) \qquad x : A, y : B, w : \mathrm{Id}_A(a, b) \vdash C(x, y, w)\ \text{type} \qquad x : A \vdash e : C(x, x, \mathrm{refl}_x)}{\mathrm{rec}_{\mathrm{Id}_A}(\lambda x.e, a, b, p) : C(a, b, p)}$$

$$\mathrm{rec}_{\mathrm{Id}_A}(\lambda x.e, a, a, \mathrm{refl}_a) \equiv e[a/x] : C(a, a, \mathrm{refl}_a)$$

Universes

We have a judgment $\vdash A$ type, how can we get a judgment $\vdash A : \mathit{Type}$?

Universes

We have a judgment $\vdash A$ type, how can we get a judgment $\vdash A : \mathit{Type}$?

We want to have a universe = a type whose elements are types

For reasons we cannot have $\mathit{Type} : \mathit{Type}$

Universes

We have a judgment $\vdash A$ type, how can we get a judgment $\vdash A : \mathit{Type}$?

We want to have a universe = a type whose elements are types

For reasons we cannot have $\mathit{Type} : \mathit{Type}$

We want to have a *hierarchy* of universes

$$\mathit{Type}_0 : \mathit{Type}_1 : \mathit{Type}_2 : \ldots$$

Universes

$$\mathbb{N}, \mathit{Bool}, \mathbb{N} \to \mathbb{N} \times \mathbb{N} : \mathit{Type}_0$$
$$\mathit{Type}_0, \mathbb{N} \to \mathit{Type}_0, \mathit{Type}_0 + \mathit{Type}_0 : \mathit{Type}_1$$
$$\ldots\ldots$$
$$\ldots : \mathit{Type}_\alpha$$
$$\ldots : \mathit{Type}_{\alpha+1}$$

Universes are indexed by ordinals/levels; we keep it abstract.

Universes and induction principles

$$\frac{n : \mathbb{N} \qquad x : \mathbb{N} \vdash C(x)\ \text{type} \qquad e_0 : C(0) \qquad n : \mathbb{N}, x : C(n) \vdash e_S : C(S(n))}{\mathrm{rec}_{\mathbb{N}}(e_0, e_S, n) : C(n)}$$

Universes and induction principles

$$\frac{n : \mathbb{N} \qquad x : \mathbb{N} \vdash C(x)\ \text{type} \qquad e_0 : C(0) \qquad n : \mathbb{N}, x : C(n) \vdash e_S : C(S(n))}{\text{rec}_{\mathbb{N}}(e_0, e_S, n) : C(n)}$$

$$\text{rec}_{\mathbb{N}} : \prod_{C:\mathbb{N}\to \mathit{Type}} C(0) \to (\prod_{x:\mathbb{N}} C(x) \to C(S(x))) \to \prod_{x:\mathbb{N}} C(x)$$

Universes and induction principles

$$\frac{n : \mathbb{N} \qquad x : \mathbb{N} \vdash C(x)\ \text{type} \qquad e_0 : C(0) \qquad n : \mathbb{N}, x : C(n) \vdash e_S : C(S(n))}{\text{rec}_{\mathbb{N}}(e_0, e_S, n) : C(n)}$$

$$\text{rec}_{\mathbb{N}} : \prod_{C:\mathbb{N}\to \mathit{Type}} C(0) \to \left(\prod_{x:\mathbb{N}} C(x) \to C(S(x))\right) \to \prod_{x:\mathbb{N}} C(x)$$

```
Nat.rec.{u} {motive : Nat → Sort u}
      (zero : motive Nat.zero)
      (succ : (n : Nat) → motive n → motive n.succ)
      (t : Nat) :
  motive t
```

Not Covered: Lean

- Tactics for writing down proofs
- The universe *Prop* of propositions, impredicativity ($\prod_{P:Prop}(P \to P) : Prop$)
- Quotient types
- Axioms and classical reasoning
- Modules, libraries, etc

Not Covered: Lean

- Tactics for writing down proofs
- The universe *Prop* of propositions, impredicativity ($\prod_{P:Prop}(P \to P) : Prop$)
- Quotient types
- Axioms and classical reasoning
- Modules, libraries, etc

Thank you for listening!