

Understanding Dependent Type Theory

David McAllester

UIUC FM/PL/CE Seminar

October 14, 2026

TakeAways

- ▶ The equivalence of Lean and ZFC.
- ▶ The validity of model theoretic (semantic) specifications of formal systems.
- ▶ A model theoretic (set semantics) specification of dependent type theory.
- ▶ Understanding the natural language of mathematicians — what do mathematicians mean by “isomorphic”, “natural”, “canonical”, “invariant”, “classification”, “symmetry”, “data”, “dual”, and “representation”.

Model Theory: Sorted First Order Logic

- ▶ A sorted first order language is defined by a set of constant, function and predicate symbols each assigned a type involving sorts.
- ▶ For example, if we have vectors and scalars then scalar-vector product $*$ is assigned the type

$$* : (scalar \times vector) \rightarrow vector$$

- ▶ A **model** of a sorted first order language assigns a set to each sort and a meaning to each constant, function and predicate consistent with the assigned types and satisfying certain axioms.

Groups

A group is a model of a first order language with a single sort (the group elements) and which assigns meanings to the constant id the function inv and the binary operations op satisfying the group axioms.

Properties of groups can be derived using the inference rules of first order logic.

Soundness and Completeness

- ▶ B follows from A (semantically) if every model of A (every model making A true) is a model of B .
- ▶ A set of inference rules is **sound** if everything provable from A follows semantically from A .
- ▶ A set of inference rules is **complete** if everything that follows from A is also provable from A .
- ▶ The inference rules of first order logic are both sound and complete.
- ▶ This implies that any first order formula that holds for all groups can be proved using the inference rules of first order logic.

Higher Order Logic (HOL)

Higher order logic allows quantification over functions and predicates. For example, we can write an induction axiom:

$$\forall P: \text{int} \rightarrow \text{bool} \quad (P(0) \wedge \forall x (P(x) \rightarrow P(s(x)))) \rightarrow \forall x P(x)$$

This axiom is false in any model contains an element not reachable from 0.

The second order Peano axioms have only one model (up to isomorphism). No first order axioms have this property.

The Necessary Incompleteness of HOL

- ▶ Gödel's incompleteness theorems imply that any system of sound inference rules for higher order logic must be incomplete.
- ▶ In what sense then is the “logic” of HOL well defined?

TakeAway I: The Equivalence of Lean and ZFC.

- ▶ The equivalence of Lean and ZFC^+ (ZFC plus inaccessible cardinals) was shown in Mario Carneiro's master's thesis (2019).
- ▶ Platonic mathematics is “captured” by ZFC^+ and hence any formal foundation acceptable to mathematicians must be ZFC^+ equivalent.

TakeAway II: Validity of Model Theoretic Specification

- ▶ For any logic defined purely by its model theory a sound and ZFC^+ -complete inference system can be constructed by formally defining the model theory (in Lean or ZFC^+) and proving (in Lean or ZFC^+) semantic entailments.
- ▶ This is just what happens when a mathematician defines the meaning of notation and then makes “Platonic” (ZFC^+) arguments.

The Type “group”: From HOL to Dependent Type Theory

The type “group” is not expressible as a type in HOL for two reasons.

- ▶ First, a simple type for the constants and functions are expressed over a particular sort — we cannot write a type of the form “a tuple whose first component is a sort and ...”.
- ▶ Second, simple types do not support incorporating axioms into a type.

TakeAway III: The Set Model of Dependent Types

- ▶ We write $s : \text{set}$ to declare a variable ranging over sets (sorts).
- ▶ We write $\Sigma(x:\sigma)\tau[x]$ for the set (or class) of pairs $\langle x, y \rangle$ such that $x:\sigma$ and $y:\tau[x]$.
- ▶ We write $\Pi(x:\sigma)\tau[x]$ for the set (or class) of functions (or functors) f such that for $x:\sigma$, we have that $f(x)$ has type $\tau[x]$.
- ▶ Refinement types have the form $\{x:\sigma \mid \Phi[x]\}$. The elements of this type are the elements x of the set (or class) σ such that $\Phi[x]$ holds.

Groups

- ▶ We formalize groups using $\text{group} : \text{set} \rightarrow \text{set}$
- ▶ For any set (sort) s and for $G:\text{group}(s)$ we have that G is a triple $(\text{op}, \text{inv}, \text{id})$ where these have the specified types over the sort s and satisfy the group axioms.
- ▶ In practice we can define $G:\text{group}(s)$ to be an object with named attributes op , inv , and id .
- ▶ Each group $G:\text{group}(s)$ is then a model of the first order language of groups giving meanings to constant and function symbols (the attributes of the object) over the sort s .

A Note on Refinement Types

- ▶ In Coq/Rocq refinement types are represented by the dependent pair type $\Sigma x:\sigma. \Phi[x]$ under the “propositions as types” doctrine, where the formula $\Phi[x]$ is a type whose elements are “proof terms”.
- ▶ In Lean this is modified so that all proofs of the same formula are taken to be equal. In this case an element of $\Sigma(y:\sigma)\Phi[y]$ can be viewed as a pair $(y, \Phi[y])$ — a pair of a value y and a formula y satisfies.
- ▶ In LEAN if the formulas $\Phi[y]$ and $\Psi[y]$ are not the same then $\Sigma(y:\sigma)\Phi[y]$ and $\Sigma(y:\sigma)\Psi[y]$ are disjoint. However, refinement types $\{y|\Phi[y]\}$ and $\{y|\Psi[y]\}$ can intersect.
- ▶ Refinement types occur in other systems such as PVS, NuPRL, and Liquid Haskell.

A Note on Polymorphism

Set arguments can typically be inferred.

Consider a polymorphic `cons` function.

$$\text{cons} : \Pi(\alpha : \text{set})(\alpha \times \text{list}(\alpha)) \rightarrow \text{list}(\alpha)$$

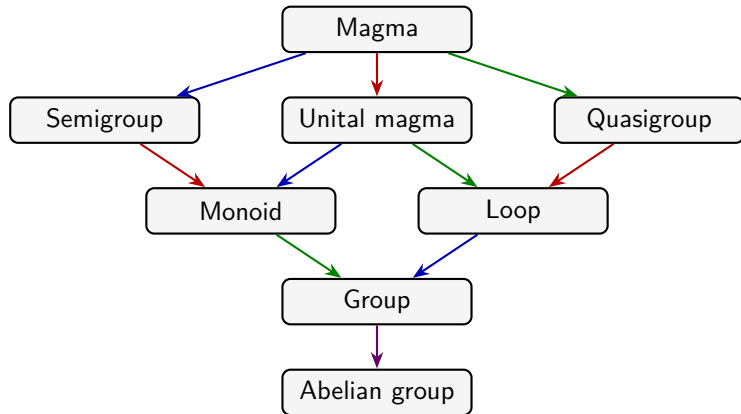
Although `cons` takes three arguments, the type argument (the sort argument) can be inferred from the types of the other arguments.

In ML or Haskell one writes `cons(x, y)` rather than `cons(s, x, y)`.

In these slides I will write type arguments as subscripts and write `conss(x, y)` instead of `cons(s, x, y)` or `cons(x, y)`.

I will also write `G:groups` where the sort `s` is part of the type of `G`.

Mathematical Ontology



→ associativity → identity → divisibility → commutativity

Each node in this graph is a “concept”.

Concepts

We define a “concept” to be **any well-typed** type constructor $A: \text{set} \rightarrow \text{set}$.

An instance X of concept A has type $X: A_s$

For example we have $G: \text{group}_s$

Isomorphisms

For any concept A we define an A -isomorphism from $X:A_s$ to $Y:A_w$ to be a bijection $f:s \rightarrow w$ which transports X to Y written as $T_{A_s}(f, X) = Y$.

For any type τ_s , value $x:\tau_s$, and bijection $f:s \rightarrow w$ the transport $T_{\tau_s}(f, x)$ is defined by structural induction on the type expression τ_s .

For the type group_s , object $G:\text{group}_s$ and f a bijection $s \rightarrow w$ we can define $T_{\text{group}_s}(f, G)$ by defining the transport of the attributes of G .

$$T_s(f, G.\text{id}) = f(G.\text{id})$$

$$T_{s \rightarrow s}(f, G.\text{inv}) = \lambda(x : w) f(G.\text{inv}(f^{-1}(x)))$$

$$T_{(s \times s) \rightarrow s}(f, G.\text{op}) = \lambda(x : w, y : w) f(G.\text{op}(f^{-1}(x), f^{-1}(y)))$$

Predicate Isomorphism Theorem

For a concept A and predicate $P: \prod(\alpha: \text{set}) A_\alpha \rightarrow \text{bool}$

and for $X: A_s$ A -isomorphic to $Y: A_w$

we have $P_s(X)$ iff $P_w(Y)$.

Functors

A functor from concept A to concept B is a function

$$F: \prod (\alpha: \text{set}, X: A_\alpha) B_{\tau_\alpha(X)}$$

For $X: A_s$ we have $F_s(X): B_{\tau_s(X)}$

Functor Isomorphism Theorem

For concepts A and B and functor F_s from A_s to $B_{\tau_s(A)}$.

if $X: A_s$ A -isomorphic to $Y: A_w$

then $F_s(X)$ is B -isomorphic to $F_w(Y)$.

Functor Covariance Theorem

For concepts A and B and functor F_s from A_s to $B_{\tau_s(A)}$.

and f an A -isomorphism from $X:A_s$ to $Y:A_w$,

$\lambda(x : \tau_s(A)) T_{\tau_s(A)}(f, x)$ is a B -isomorphism from $F_s(X)$ to $F_w(Y)$.

Vector Spaces

Vector spaces have two sorts — scalars and vectors.

We can define `VectorSpace` such that for

$$X:\text{VectorSpace}(F)(s)$$

we have that X has attributes $+$ and $\times$ (vector addition and scalar multiplication) and is a vector space over the field F where s is the sort for the vectors.

We also have that $\text{VectorSpace}(F):\text{set} \rightarrow \text{set}$ is a concept and we can write $X:\text{VectorSpace}(F)_s$.

TakeAway IV: The Language used by Mathematicians

The natural language used by by mathematicians reflects the structure of dependent type theory.

We have already discussed the type-theoretic treatment of **isomorphism**.

A **symmetry** of an object $x: C_s$ is just an isomorphism $f: s \rightarrow s$ of x with itself. The rotational symmetry of a circle is one example. Symmetry is of central importance to fundamental physics.

The term **natural** or equivalently **canonical** is defined in terms of symmetry. An object $Y: B_s$ is a natural or canonical property of $X: A_s$ if every symmetry of X is also a symmetry of Y . If Y is not canonical for A then no functor on A can map X to Y .

There is no natural point on a geometric circle and no natural isomorphism between a vector space and its dual.

TakeAway IV: The Language used by Mathematicians

An **invariant** on a category A is a simply functor F from A to B . Invariants are used to **classify** objects in a category: — if $F_s(X)$ is not B -isomorphic to $F_w(Y)$ then X and Y are not isomorphic.

A **duality** between two categories A and B is a pair of functors $F_s: A_s \rightarrow B_s$ and $G_s: B_s \rightarrow A_s$ which are inverses of each other. This allows different languages to be used on “the same” object.

If A and B are dual we say they are **cryptomorphic** or have the same **data**.

For any concept A and object $X: A_\sigma$ where σ is a type other than a set variable, we say that X is an **A-representation**. For example groups are often represented by sets of permutation or non-singular matrices.

TakeAways

- ▶ The equivalence of Lean and ZFC.
- ▶ The validity of model theoretic (semantic) specifications of formal systems.
- ▶ A model theoretic (set semantics) specification of dependent type theory.
- ▶ Understanding the natural language of mathematicians — what do mathematicians mean by “isomorphic”, “natural”, “canonical”, “invariant”, “classification”, “symmetry”, “data”, “dual”, and “representation”.