

# Formalization of ZF set theory and modal logic in Lean

Shuhao Song

October 18, 2024

My graduate thesis focuses on the formalization of ZF set theory, particularly concerning large cardinals beyond choice, such as Reinhardt and Berkeley cardinals. I will discuss some details about my formalization plan.

Inspired by Zhang Zhiqing, I completed the formalization of Gödel's ontological proof in Lean. Using a trick with typeclass synthesizing, Kripke's semantics can be easily formalized in Lean, so that all pre-existing tactics can be used. Some modification to the typeclass synthesizing process in Lean were necessary to complete this formalization, and I will present these in my talk.

# Models of ZFC in Lean

In Lean we can define an inductive type

```
inductive PSet : Type (u + 1)
  | mk ( $\alpha$  : Type u) (A :  $\alpha \rightarrow$  PSet) : PSet
```

This means, for any type  $\alpha$  and a family of pre-set  $A : \alpha \rightarrow \text{PSet}$ , we can collect them to construct a new pre-set, and every pre-set is constructed in this way. We define the membership relation as  $x \in \text{mk } \alpha \ A$  iff  $x$  is in the image of  $A$ .

Pre-sets are not exactly sets: different  $\alpha$  and  $A$  can give the same set (in set-theoretic sense). For example,  $\alpha = \{1\}, A(1) = \emptyset$  and  $\alpha = \{1, 2\}, A(1) = A(2) = \emptyset$  builds same set, as  $\{\emptyset\} = \{\emptyset, \emptyset\}$ .

So we need to take a quotient in Lean. We define the extensionality relation recursively:

```
def Equiv : PSet  $\rightarrow$  PSet  $\rightarrow$  Prop
  |  $\langle \_ , A \rangle, \langle \_ , B \rangle \Rightarrow$ 
    ( $\forall a, \exists b, \text{Equiv } (A a) (B b)$ )  $\wedge$  ( $\forall b, \exists a, \text{Equiv } (A a) (B b)$ )
```

Then define **ZFSet** to be the quotient of type **PSet** under the equivalence relation **Equiv**.

```
def ZFSet : Type (u + 1) := Quotient PSet.setoid.{u}
```

# Choiceless ZFSet

With axiom of choice, we can prove ZFSet is a model of second-order ZFC easily. Here, “second-order ZFC” means the separation and replacement axiom is quantified over all subsets of ZFSet and all functions  $\text{ZFSet} \rightarrow \text{ZFSet}$ :

$$\begin{aligned} & \forall S \forall x \exists y (z \in y \leftrightarrow z \in x \wedge S(x)) \quad (\text{Sep}) \\ & \forall f \forall x \exists y (z \in y \leftrightarrow \exists w (w \in x \wedge f(w) = z)) \quad (\text{Rep}) \end{aligned}$$

We need to use the axiom of choice in the proof of replacement axiom. The article<sup>1</sup> gave another solution to make ZFSet a model of second-order ZF without choice. They added some axioms to PSet:

**axiom**  $\gamma : \text{PSet} \rightarrow \text{PSet}$

**axiom**  $\gamma_{\text{ext}} : \forall x y, \text{PSet.Equiv } x y \rightarrow \gamma x = \gamma y$

**axiom**  $\gamma_{\text{equiv}} : \forall x, \text{PSet.Equiv } (\gamma x) x$

---

<sup>1</sup>Dominik Kirst and Gert Smolka. “Large model constructions for second-order ZF in dependent type theory”. In: Proceedings of the 7th ACM SIGPLAN International Conference on Certified Programs and Proofs. CPP 2018. Los Angeles, CA, USA: Association for Computing Machinery, 2018, pp. 228–239. ISBN: 9781450355865.

# Reflection Argument

The ZFSet is only one model of ZF. So is it enough representative? If we proved some theorems of ZFSet, then would it be true in other models of ZF? We can solve this problem using reflection in set theory. Firstly, ZFSet is a model of **second-order** ZF, and intuitionistically, we can interpret types in Coq or Lean as sets,  $\alpha \rightarrow \beta$  as the set of functions from  $\alpha$  to  $\beta$ . So by Zermelo's categoricity theorem,  $\langle \text{ZFSet}, \epsilon \rangle$  is isomorphic to certain  $V_\kappa$  for some (strongly, the same below) inaccessible cardinal  $\kappa$ . Note that the categoricity theorem is already formalized in Coq<sup>2</sup>. Moreover, inaccessibility is a subtle notion without choice, and here we refer to  $\nu$ -inaccessibility in the literature<sup>3</sup>.

---

<sup>2</sup>Dominik Kirst and Gert Smolka. "Categoricity Results and Large Model Constructions for Second-Order ZF in Dependent Type Theory". In: *Journal of Automated Reasoning* 63.2 (2019), pp. 415–438. ISSN: 1573-0670.

<sup>3</sup>Andreas Blass, Ioanna M. Dimitriou, and Benedikt Löwe. "Inaccessible cardinals without the axiom of choice". In: *Fundamenta Mathematicae* 194 (2003), pp. 179–189. 

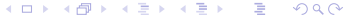
# Reflection Argument

Assuming we have proven certain first-order sentence  $\varphi$  in  $\langle \text{ZFSet}, \in \rangle$ . Let  $V$  be the universe (or model-theoretically, a certain model of ZF), we can prove that if  $V$  satisfies “Ord is Mahlo”, then  $V$  satisfies  $\varphi$ . So, by Gödel’s completeness theorem, we have proven  $\varphi$  in  $\text{ZF} + \text{Ord is Mahlo}$ .

In detail, given a proper class  $R$ , we can prove that the ordinals  $\kappa$  with  $\langle V_\kappa, \in, R \cap V_\kappa \rangle <_n \langle V, \in, R \rangle$  forms a closed unbounded proper class, where  $<_n$  means  $\Sigma_n$ -elementary embedding. So if Ord is Mahlo, which means the class of inaccessible cardinals intersects with every club class, there should exist one  $\kappa$  such that  $V_\kappa$  reflects  $V$ . So if  $\neg\varphi$  is true in  $V$ , it must be true in some  $V_\kappa$ , then we interpret the type-theoretic universe  $\text{Type } 0, \dots, \text{Type } n, \dots$  as  $V_\kappa, V_{\kappa_1}, V_{\kappa_2}, \dots$ , where  $\kappa < \kappa_1 < \dots$  are inaccessible cardinals, so we can prove  $\varphi$  is true in  $V_\kappa$  using set-interpretation of type theory, leading a contradiction.

Note that the reflection argument fails in CZF and IZF.<sup>4</sup>

---

<sup>4</sup>Reflection principle for intuitionistic Zermelo–Fraenkel? <https://mathoverflow.net/questions/319179/reflection-principle-for-intuitionistic-zermelo-fraenkel>. 

# Types and Sets

What is the difference of **Type** and  $V_\kappa$ ?

Adding the following axioms can make **Type** behaves more similar to sets in ZF set theory.

```
universe u
axiom typeMk :  $\forall \{a\}, (a \rightarrow \mathbf{Type} \ u) \rightarrow \mathbf{Type} \ u$ 
def ExtEq {a  $\beta$ } (f : a  $\rightarrow$  Type u) (g :  $\beta \rightarrow$  Type u) :=
  ( $\forall x : a, \exists y : \beta, f \ x = g \ y$ )  $\wedge$   $\forall y : \beta, \exists x : a, f \ x = g \ y$ 
axiom type_ext :  $\forall \{a \beta\} (f : a \rightarrow$  Type u) (g :  $\beta \rightarrow$  Type u),
  ExtEq f g  $\leftrightarrow$  typeMk f = typeMk g
axiom type_rec :  $\forall \{p : \mathbf{Type} \ u \rightarrow \mathbf{Prop}\}$ 
  ( $\_ : \forall \{a\} \{f : a \rightarrow$  Type u}, ( $\forall x, p (f \ x)) \rightarrow p (typeMk \ f)$ )
  (x : Type u), p x
```

# Definite Description

With axioms of ZF, we can prove the existence of certain set, but we can't write down the term for the set. For example, we can prove the proposition  $\exists x \forall y (y \notin x)$ , but we can't get a term  $\emptyset : \text{ZFSet}$ . So we need a new axiom in type theory for this. It is called “definite description”: if you can use certain property to uniquely specify one object, then you can obtain the object. In Lean we write

```
axiom definiteDescription :  
  { $\alpha$  : Type*}  $\rightarrow$  Nonempty  $\alpha$   $\rightarrow$  Subsingleton  $\alpha$   $\rightarrow$   $\alpha$ 
```

A type  $\alpha$  in Lean is called subsingleton if all elements in  $\alpha$  are equal, that is,  $\forall x y : \alpha, x = y$ . So, a nonempty subsingleton type can only have exactly one element, and we can obtain this element using axiom `definiteDescription`. Note that without the assumption `Subsingleton  $\alpha$` , the axiom become the choice: you postulated a choice function  $V_\kappa \setminus \{\emptyset\}$ .



## Formalizing Kripke's frame

We formalize objects in model logic using “type with world”. For example, the truth of proposition may dependent on the world, so we use `World → Prop` instead of `Prop`.

The world can be automatically deduced from context: when we write  $P x \rightarrow \Box(Q x \wedge \Diamond(R x))$ , it means “for every world  $w$ , if  $P(x)$  is true at  $w$ , then for any  $w \rightarrow w'$ ,  $Q(x)$  is true and there exists  $w' \rightarrow w''$  such that  $R(x)$  is true at  $w''$ ”.

In our formalization, we can directly write the proposition  $P x \rightarrow \Box(Q x \wedge \Diamond(R x))$  “as-is” in Lean, and Lean can automatically introduce the three worlds  $w$ ,  $w'$ ,  $w''$ , and know  $P(x)$  is at  $w$ ,  $Q(x)$  and  $R(x)$  is at  $w'$  and  $w''$  correspondingly. So how can we achieve this? We use implicit lambda feature to automatically introduce variables, and use typeclass synthesize to make  $P, Q, R$  know which world are they in.

# Formalizing Necessary and Possible modality

The “elaborate” step turns a syntax object to an expression in Lean kernel, which is a semantical object. When we write  $\Box \forall x, P x$ , the elaborator knows  $\forall x, P x$  is a **Prop** and  $\Box$  needs a **&Prop**, where we use symbol  $\&\alpha$  to denote  $[w : \text{World}] \rightarrow \alpha$ .  $\Box$  is just a notation for definition Necessary.

```
def Necessary (p : &Prop) :=  $\forall w', \text{Accessible } w w' \rightarrow p@.w'$ 
def Possible (p : &Prop) :=  $\exists w', \text{Accessible } w w' \wedge p@.w'$ 
notation "□" p:50 => Necessary p
notation "◇" p:50 => Possible p
```

Because  $[w : \text{World}]$  is an implicit parameter (it’s a typeclass, which can be automatically deduced from the environment), it would be automatically introduced to unify the type **&Prop** and **Prop**. But we also need to deduce the world from context, this is done using typeclass synthesizing. The type `World` is a type class, and every object of `World` in the current proof context is an instance of it. The later introduced instance (`world`) gets higher priority, so  $P x \wedge \Box Q x$  will use  $w'$  (introduced in  $\Box$ ) instead  $w$  as the “current” world, which agrees with our common understanding.

## Modification to Lean

We modified the typeclass synthesizing process to make this possible. When elaborating  $\diamond \exists x, P x$ , the type of  $x$  must be deduced; it should be `Object`. But in Lean, the type of  $x$  will be deduced as  $x : @?m.49388 P w \dagger$  firstly, which means, the type of  $x$  may depends on  $P$  and newly introduced world  $w \dagger$  in  $\diamond$ . Here,

`?m.49388 : {P : Property} → [w : World] → Sort ?u.49374` is a “metavariable”, which means an unknown term. So the problem occurs: it should be `fun P w => Object w`, so `@?m.49388 P w \dagger` will be the correct type `Object w \dagger`, but  $w$  is in the parameter of function, and in Lean, if an instance is at the position of function parameter it will be ignored in the typeclass resolution. We modified the Lean source code to consider this case.

# Proof of existence of God

Some literature<sup>5,6</sup> defined the type of formulas and models, and a satisfaction relation:

`forces_form : ∀ {A : Type} (M : model A), form → A → Prop`

This method is suitable to prove meta-properties (such as completeness and soundness), but it could be hard to prove inner theorems, as you have to translate between the satisfaction of formula and its semantical expansion. For example, you need to translate between

`forces_form M (form.and p q) w` and

`forces_form M p w ∧ forces_form M q w`. Maybe it can be done with automatically `@[simp]` attribute. Our method make us able to reuse many pre-existing tactics in Lean to reason in modal logic, which makes our proof easier (only half a day).

---

<sup>5</sup>Bruno Bentzen. “A Henkin-Style Completeness Proof for the Modal Logic S5”. In: *Logic and Argumentation: 4th International Conference, CLAR 2021, Hangzhou, China, October 20–22, 2021, Proceedings*. Hangzhou, China: Springer-Verlag, 2021, pp. 459–467. ISBN: 978-3-030-89390-3.

<sup>6</sup>Huayu Guo, Dongheng Chen, and Bruno Bentzen. Verified completeness in Henkin-style for intuitionistic propositional logic. 2023. arXiv: 2310.01916 [cs.LO].

# Proof of existence of God

Example code (uses lots of Lean tactic)

```
theorem essential_God :  $\forall \{x\}, \text{God } x \rightarrow \text{Essential God } x := \text{by}$   
  intro x hx  
  refine <hx, fun Q hQ => ?_>  
  have pos_Q :  $\square \text{Positive } Q := \text{by}$   
    apply positive_necessary  
    by_contra h  
    rw [ $\leftarrow$  positive_or_not] at h  
    exact hx h hQ  
  revert pos_Q  
  apply necessary_mp  
  exact fun w' _ pos y hy => hy pos
```

An example of term-style proof, may be harder to read than tactic-style proof above.

```
theorem God_exists :  $\square \exists x, \text{God } x :=$   
  possible_necessary_imp <|  
    possible_mp  
    (necessarize necessarily_exists_God)  
    (positive_possible_exists positive_God)
```