Fall 2025 Directed Reading Program: Lean 4 and Forcing

Mentor: Bryan Wang Peng Jun

Mentees: Mason Shuttleworth & Iris Sung

1. Introduction

In this report, we detail the basics of Lean 4 and its foundations as a type theoretic proof assistant in Section 2. In Section 3, we briefly recount various method of forcing in set theory. Section 4 covers the efforts of the Flypitch project, and how they approached formalizing forcing in Lean 3. The methods used in Flypitch could be adapted in an effort to formalize forcing in Lean 4 and contribute to Mathlib, Lean's expansive library.

2. Lean 4 Basics

The primary textbook we utilized for an introduction into Lean 4 was "Mathematics in Lean" (MIL) written by Jeremy Avigad and Patrick Massot. The textbook is designed to be read interactively inside of the VS Code editor, as it comes filled with plenty of examples and exercises.

2.1. **Dependent type theory & Lean.** Lean is a tool to formalize complex mathematical expressions, in the formal language of *dependent type theory*. Every expression has a *type*, and each type lives in a hierarchy of universes, `Type u` for $u = 0, 1, 2, \ldots$. [AM25, Mat20]. Lean distinguishes between normal types, such as `Nat` for the natural numbers or $\mathbb{R}$ for the reals, and the special type `Prop` for propositions, or mathematical statements [AM25].

Definition 2.1. A *type* in Lean is any of the form α `: Type u` for some universe u.

Definition 2.2. A *proposition* in Lean is a term `P : Prop`.

Definition 2.3. A *proof* of a proposition P is a term `h : P`.

Example 2.4. The statement $2 + 2 = 4$ can be represented as a term of type `Prop`, and a proof is a term of that type. In Lean 4 syntax (where any line beginning with `--` gets ignored by Lean),

```
-- A proposition:
#check (2 + 2 = 4)
-- : Prop
```

```
--A proof:
theorem easy : 2 + 2 = 4 :=
  rfl

#check easy
-- : 2 + 2 = 4
```

In this example, `rfl` is a proof term (where `rfl` is short for reflexivity). It proves goals of the form `t = t` and goals which reduce to that form after computation. More generally, a theorem in Lean is a constant whose type is a proposition. The text of the proof is just an expression, creating a term of that type [AM25].

Definition 2.5. Let P be a proposition in Lean. A term `h : P` is a *proof term* of P. If such a term exists for P, and Lean accepts it, then P is considered *provable* in Lean.

MIL mentions two equivalent ways of writing proofs: as explicit proof terms, and by using tactics that build such terms step by step. MIL mainly focuses on the tactics approach [AM25].

Example 2.6. For example,

```
theorem mul_even_of_even
(m n : Nat) (hn : \exists k, n = 2 * k) :
\exists k, m * n = 2 * k := by
rcases hn with ⟨k, hk⟩
refine ⟨m * k, ?_⟩
simp [hk, Nat.mul_assoc, Nat.mul_comm]
```

Here, `rcases` and `refine` are tactics, which can be used to construct a proof term whose type is the desired conclusion.

2.2. **Sets in Lean's type theory.** Although Lean is based on types rather than sets, sets and many set-theoretic notions have been formalized in Lean and are present in Mathlib [Mat20]. A key observation in MIL is that a subset of a type α can be represented as `\alpha \to Prop`. For s with type `s : \alpha \to Prop` and x with type `x : \alpha`, we write `s x` in Lean to mean the proposition "$x \in s$" [AM25].

Definition 2.7. For a type α, a *set of elements of* α is a predicate `s: \alpha \to Prop`. Lean's notation `Set \alpha` means, by definition, `\alpha \to Prop` [AM25].

One can find more elaborate definitions in Mathlib, accompanied by a plethora of lemmas and such with the usual notions of $\cup$, $\cap$, and so on [AM25, Mat20]. In particular,

Lemma 2.8. *Let* `s,t,u : Set \alpha`. *If* $s \subseteq t$, *then*

$$s \cap u \subseteq t \cap u.$$

In Lean, this corresponds to

```
open Set

lemma inter_subset_inter {\alpha : Type u}
{s t u : Set \alpha} (h : s \subseteq t) :
s \cap u \subseteq t \cap u := by
intro x hx
exact ⟨h hx.1, hx.2⟩
```

[Mat20, AM25]

3. Forcing

Forcing is a set theoretic method by which a ground model (a transitive model of set theory) is extended to a larger transitive model of set theory, called a generic extension by a set called a generic set. We pick forcing conditions in the ground model to represent the generic set, and use these to ascertain truths of the generic extension [J02].

We typically denote the transitive model by M, the generic set by G and the generic extension by $M[G]$.

The original method of forcing, known as Cohen forcing (after Paul Cohen), was used to prove the independence of the Continuum Hypothesis (CH) from ZFC. This is the primary focus of the following parts. Further forms of forcing, including iterated forcing, Prikry-type forcing, and many more are extensions of this idea and can be used to show the further independence of certain axioms from chosen models of set theory.

In the subsequent parts, our model M is a transitive model of ZFC, and we define a generic set G, such that $M[G]$ is a transitive model of ZFC, for which CH fails. Stating that a M is a transitive model of ZFC means that the axioms of ZFC all hold in our model.

3.1. An Introduction to Forcing. [J02]

Definition 3.1 (Notion of forcing)**.** We consider a partially ordered set, or poset, $(P, <)$ in our ground model M, which we call a notion of forcing, with elements of P called forcing conditions. Definitionally, we say p is stronger than q if $p < q$ and that p and q are compatible if there is another condition r such that $r \leq p$ and $r \leq q$. Otherwise p and q are incompatible. P is used to represent the both the set, as well as the notion of forcing.

Definition 3.2 (Dense)**.** We further define a set D in P to be dense if for every p in P, there is a q in D such that $q \leq p$.

Definition 3.3 (Antichain)**.** We call a set W in P an antichain if all elements are pairwise incompatible.

Definition 3.4 (Filter)**.** A nonempty set $F \subset P$ is a filter on P if

(1) $p \leq q$ and $p \in F$ implies that $q \in F$ and
(2) if $p, q \in F$ implies that there exists a condition r such that $r \leq p$ and $r \leq q$.

Definition 3.5 (Generic)**.** Finally, a set $G \subset P$ is (P-)generic over M if

(1) it is a filter on P
(2) for any dense D in P, with $D \in M$, we have a nonempty intersection of G and D.

There are two main theorems that we use to define the remaining aspects of forcing: generic models and the forcing relation.

Theorem 3.6 (Generic Model Theorem)**.** *For M a ground model, P a notion of forcing and G generic over P, we have that there exists a transitive model $M[G]$, which we call the generic extension, such that*

(1) *$M[G]$ is a model of ZFC*
(2) *$M \subset M[G]$ and G is an element of the generic extension*
(3) *ordinals in the ground model and the generic extension are equivalent*
(4) *$M[G]$ is contained in any transitive model of ZFC which contains the ground model as a subset, and the generic set as an element.*

For the forcing theorem, we take a few additional definitions.

Definition 3.7 (Names)**.** A P-name is a collection of pairs (y, p) with y a P-name and $p \in P$. More generally, we say that elements of the generic extension have names in the ground model which describe the corresponding element of $M[G]$. We denote by M^P the class of all names in M.

Definition 3.8 (Forcing Relation)**.** We say σ is a sentence in the forcing language if $p \Vdash \sigma$ (or p forces σ). Then, the forcing relation relates conditions to sentences. Further, if σ implies some other ϕ, then $p \Vdash \sigma$ implies $p \Vdash \phi$.

Theorem 3.9 (Forcing Theorem)**.** *For P a notion of forcing in ground model M. For any sentence, we say for every $G \subset P$ generic over M, that $M[G] \Vdash \sigma$ if and only if there is a condition p in G such that $p \Vdash \sigma$.*

We conclude with a few properties of the forcing relation.

Proposition 3.10 (Properties of Forcing)**.** *For P notion of forcing in M,*

(1) $p \Vdash \phi$, $q \leq p$ *implies* $q \Vdash \phi$
(2) $p \Vdash \phi$ *implies* $\neg(p \Vdash \neg\phi)$
(3) *For every p, there exists a $q \leq p$ such that $q \Vdash \phi$ or $q \Vdash \neg\phi$. We say that q decides ϕ.*

3.2. **Forcing by Boolean Algebras.** [HD20] Set theoretically, boolean algebras form an equivalent, but more challenging, argument for forcing than an argument by P-names. However, in the type-theoretic basis of Lean, boolean algebras are more easily encoded and manipulated. As such, the Flypitch project follows this argument [HD20]. To do so, we fix a complete Boolean algebra, $\mathbb{B}$, that belongs to M, our ground model, as above. We can informally define a complete Boolean algebra as being such that every subset of $\mathbb{B}$ has a supremum and infimum, which allows us to fix a partial ordering, as we do when considering transitive models above. Then, we consider similar constructions as above.

Definition 3.11. We define our generic extension, $M[G]$ as $\{x^G : x \in M^{\mathbb{B}}\}$.

Although this may appear to be an identical construction, set theoretically, it is simpler to formalize by complete boolean algebras, and as such, Flypitch adopts the notation of boolean algebra.

4. The Flypitch Project

The Flypitch project, carried out by Han and van Doorn, sought to give a fully formal proof of the independence of the continuum hypothesis from ZFC in the Lean theorem prover [HD20]. Their 2020 paper describes the final stage of the project, where they obtain Boolean-valued models of ZFC in with the continuum hypothesis has truth value $\top$ and $\bot$. Their formalization was done entirely in Lean 3 rather than the newer release of Lean 4, and builds on top of Mathlib. The project is publicly available as a Lean repository [HD20].

4.1. **Flypitch's Approach to Formalizing Forcing.** [HD20]

The primary difficulty with formalizing forcing in Lean is the need to reconcile set theoretic constructions with the type theoretic basis of Lean. Flypitch does so by following the Boolean algebra method detailed above and using a universe of types. This is done in Lean by using expressions, such as Type `u` and Type `u+1` to denote the relevant universe. In general, in Lean, types are associated with universes, indexed by natural numbers, such that smaller types are elements within the next larger type. Fixing a Type `u` allows for considering models at different set theoretic universes. As such, the forcing takes place in the universe of Type `u`, with elements being the Boolean valued models $\mathbb{B}$. Further, they label the set of $\mathbb{B}$-names (as described above) as bSets, of Type `u+1`. This is particularly useful in more complex constructions, such as for proving the consistency of CH with ZFC, though the consistency of ¬CH with ZFC is a simpler proof [HD20].

5. Further Directions

We would like to continue our work by translating the Flypitch Lean 3 code to Lean 4 code, simplifying where possible. In addition, we would like to consider formalizing using the initial, more traditional approach to forcing introduced in Section 3.1. Further, we would like to formalize iterated forcing, which is an extension of Cohen forcing. Finally, we would like to consider other, more complex notions of forcing and their formalizations.

Acknowledgements

This work was done as a part of the Directed Reading Program of Harvard University for the Fall 2025 semester. We would like to thank our mentor, Bryan Wang Peng Jun, for his guidance and for providing us with a great introduction into Lean, catered toward our interests in mathematical logic and set theory. We plan to take Math 161 in the spring to further develop our proficiency with Lean 4, and to continue this work.

References

[AM25] J. Avigad, P. Massot. “Mathematics in Lean”. `https://avigad.github.io/mathematics_in_lean/mathematics_in_lean.pdf`

[C09] J. Cummings. “Iterated Forcing and Elementary Embeddings”. 2009. `https://www.math.cmu.edu/users/jcumming/papers/repaper_finished_june_2008.pdf`

[HD20] J. Han & F. Doorn. “A Formal Proof of the Independence of the Continuum Hypothesis”. In Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP ’20), January 20–21, 2020, New Orleans, LA, USA. ACM, New York, NY, USA, 14 pages. https://doi.org/10.1145/3372885. 3373826

[J02] T. Jech. “Set Theory: The Third Millennium Edition, revised and expanded”. 2002.

[Mat20] The mathlib Community. “The Lean Mathematical Library”. In: Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs. CPP 2020. New Orleans, LA, USA: Association for Computing Machinery, Jan. 2020, 367–381. DOI: 10.1145/3372885.3373824. URL: https://doi.org/10.1145/ 3372885.3373824.