A formalization of forcing and the consistency of the failure of the continuum hypothesis

Jesse Michael Han[1]
Department of Mathematics, University of Pittsburgh
`https://www.pitt.edu/~jmh288`
jessemichaelhan@gmail.com

Floris van Doorn
Department of Mathematics, University of Pittsburgh

Abstract

We describe a formalization of forcing using Boolean-valued models in the Lean 3 theorem prover, including the fundamental theorem of forcing and a deep embedding of first-order logic with a Boolean-valued soundness theorem. As an application of our framework, we specialize our construction to a Boolean completion of the Cohen poset and formally verify in the resulting model the failure of the continuum hypothesis.

2012 ACM Subject Classification Theory of computation → Logic and verification; Theory of computation → Type theory; Software and its engineering → Formal methods

Keywords and phrases Interactive theorem proving, formal verification, set theory, forcing, independence, continuum hypothesis, Boolean-valued models, Lean

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

Acknowledgements The authors would like to thank the members of the Pitt-CMU Lean group, particularly Simon Hudon, Jeremy Avigad, Mario Carneiro, and Tom Hales for their feedback and suggestions; we are also grateful to Dana Scott and John Bell for their advice and correspondence.

Introduction

The continuum hypothesis states that there are no sets strictly larger than the countable natural numbers and strictly smaller than the uncountable real numbers. It was introduced by Cantor in 1878 and was the very first problem on Hilbert's list of twenty-three outstanding problems in mathematics. Gödel proved in 1938 [?] that the continuum hypothesis was consistent with ZFC, and later conjectured that the continuum hypothesis was independent of ZFC, i.e. neither provable nor disprovable from the ZFC axioms. In 1963, Paul Cohen developed *forcing* [?], which allowed him to prove the consistency of the negation of the continuum hypothesis, and therefore complete the independence proof. For this work, which marked the beginning of modern set theory, he was awarded a Fields medal—the only one to ever be awarded for a work in mathematical logic.

The work we describe in this paper is part of the Flypitch project[2], which aims to formalize the independence of the continuum hypothesis. Our results mark a major milestone towards that goal.

Our formalization is written in the Lean 3 theorem prover. Lean is an interactive proof assistant under active development at Microsoft Research [?] [?]. It implements the Calculus of Inductive Constructions and has a similar metatheory to Coq, adding definitional proof irrelevance, quotient types, and a noncomputable choice principle. There is a well-known

[1] Corresponding author.
[2] `https://github.com/flypitch/flypitch/`

encoding of ZFC into dependent type theory with CIC, due to Aczel and Werner, which has been implemented in Lean's mathematical components library `mathlib`. The fact that Lean's metatheory is powerful enough to encode a model of ZFC already allows us to perform metatheoretic arguments about ZFC which were unavailable to e.g. Paulson [?], who went to extreme lengths to circumvent them inside Isabelle/ZF. While this was a formidable task, we content ourselves with treating ZFC as a mathematical object of study, and freely ignore the restrictions which it would impose as a foundation for the metatheory.

Indeed, our formalization makes as much use of the expressiveness of Lean's dependent type theory as possible, using constructions which are impossible or unwieldy to encode in HOL, much less ZF: Lean's ordinals and cardinals, which are defined as equivalence classes of well-ordered types, live one universe level up and play a crucial role in the forcing argument; the models of set theory we construct require as input entire universes of types; our encoding of first-order logic crucially uses parametrized inductive types to equate type-correctness with well-formedness, eliminating the need for separate well-formedness proofs.

Why Boolean-valued models? The method of forcing with Boolean-valued models was developed by Solovay and Scott (and independently, Vopěnka) in '65-'66 [?] [?] as a simplification of Cohen's method. Some of these simplifications were incorporated by Shoenfield [?] into a general theory of forcing using partial orders, and it is in this form that forcing is usually practiced. While both approaches have essentially the same mathematical content (see e.g. the discussion in Kunen [?] or Jech [?]), there are several reasons why we chose Boolean-valued models for our formalization:

- **Modularity.** The theory of forcing with Boolean-valued models cleanly splits into several components (a general theory of Boolean-valued semantics for first-order logic, a library for calculations inside complete Boolean algebras, the construction of Boolean-valued models of set theory, and the specifics of the forcing argument itself) which could be formalized in parallel and then recombined.
- **Directness.** For the purposes of an independence proof, the Boolean-valued soundness theorem eliminates the need to produce a two-valued model. This approach also bypasses any requirement for the reflection theorem/Löwenheim-Skolem theorems, Mostowski collapse, countable transitive models, or genericity considerations for filters.
- **Novelty and reusability.** As far as we were able to tell, the Boolean-valued approach to forcing has never been formalized. Furthermore, while for the purposes of an independence proof, forcing with Boolean-valued models and forcing with countable transitive models accomplish the same thing, a general library for Boolean-valued semantics of a deeply embedded logic could be used for formal verification applications outside of set theory, e.g. to formalize the Boolean-valued semantics of the stochastic λ-calculus [?].
- **Amenability to structural induction.** As with Coq, Lean is able to encode extremely complex objects and reason about their specifications using inductive types. However, the user must be careful to choose the encoding so that properties they wish to reason about are accessible by structural induction, which is the most natural mode of reasoning in the proof assistant. After observing (1) that the Aczel-Werner encoding of ZFC as an inductive type is essentially a special case of the recursive *name* construction from forcing (c.f. Section 3), and (2) that the automatically-generated induction principle for that inductive type *is* $\in$-induction, it is easy to see that this encoding can be modified to produce a Boolean-valued model of set theory where, again, $\in$-induction comes for free.

The rest of the paper is organized as follows. In Section 1 we outline the method of Boolean-valued models and sketch the forcing argument. Section 2 discusses a deep embedding of first-order logic, including a proof system, soundness and completeness theorems, and crucially Boolean-valued semantics and the Boolean-valued soundness theorem. Section 3 discusses our construction of Boolean-valued models of set theory, emphasizing the usefulness of being able to metaprogram custom tactics to simulate predicate calculus inside an arbitrary complete Boolean algebra. Section 4 describes the formalization of the forcing argument and the construction of a suitable Boolean algebra for forcing ¬CH. Section 5 describes the formalization of the Δ-system lemma, a technical result in transfinite combinatorics which ensures the preservation of cardinal inequalities. We conclude with an indication of future work towards a formal proof of the independence of the continuum hypothesis.

1 Outline of the proof

ZFC is a collection of first-order sentences in the language of a single binary predicate relation $\{\in\}$, used to axiomatize set theory. The continuum hypothesis can be written in this fashion as a first-order sentence CH. A proof of CH is a finite list of deductions starting from ZFC and ending at CH. The soundness theorem says that provability implies satisfiability, i.e. if ZFC $\vdash$ CH, then CH interpreted in any model of ZFC is true. Taking the contrapositive, we can demonstrate the unprovability (equivalently, the consistency of the negation) of CH by exhibiting a single model where CH is not true.

A model of a first-order theory T in a language L is in particular a way of assigning true or false in a coherent way to sentences in L. Modulo provable equivalence, the sentences form a Boolean algebra and "coherent" means the assignment is a Boolean algebra homomorphism (so $\wedge$ becomes meet, $\vee$ becomes join, $\forall$ becomes an indexed infimum, etc.) into $\mathbf{2} = \{\mathsf{true}, \mathsf{false}\}$. The soundness theorem ensures that this homomorphism v sends a proof $\phi \vdash \psi$ to an inequality $v(\phi) \leqslant v(\psi)$. But $\mathbf{2}$ does not really play a special role in this scheme, and may be replaced by any complete Boolean algebra $\mathbb{B}$, where the top and bottom elements $\top, \bot$ take the place of true and false. It is straightforward to extend this analogy to a $\mathbb{B}$-valued semantics for first-order logic, and in this generality, the soundness theorem now says that for any such $\mathbb{B}$, if ZFC $\vdash$ CH, then for any $\mathbb{B}$-valued structure where all the axioms of ZFC have truth-value $\top$, CH does also. Then as before, to demonstrate the consistency of the negation of CH it suffices to find just one $\mathbb{B}$ and a single $\mathbb{B}$-valued model where CH is not "true".

This is where forcing comes in. Given a universe V of set theory which contains a Boolean algebra $\mathbb{B}$, one constructs in analogy to the cumulative hierarchy a new $\mathbb{B}$-valued universe $V^{\mathbb{B}}$ of set theory, where the powerset operation is replaced by taking functions into $\mathbb{B}$. Thus, the structure of $\mathbb{B}$ informs the decisions made by $V^{\mathbb{B}}$ about what subsets, hence functions, exist among the members of $V^{\mathbb{B}}$; the real challenge lies in selecting a suitable $\mathbb{B}$ and reasoning about how its structure affects the structure of $V^{\mathbb{B}}$. While $V^{\mathbb{B}}$ may vary wildly depending on the choice of $\mathbb{B}$, the original universe V always embeds into $V^{\mathbb{B}}$ via an operation $x \mapsto \check{x}$, and while the passage of x to $\check{x}$ may not always preserve its original properties, Δ_0-properties are always preserved; in particular, $V^{\mathbb{B}}$ thinks $\check{\mathbb{N}}$ is $\mathbb{N}$.

To force the negation of the continuum hypothesis, we use the Boolean algebra of regular opens of the Cantor space $\mathbb{B} := \mathrm{RO}(2^{\aleph_2 \times \mathbb{N}})$. For each $\nu \in \aleph_2$, we associate the $\mathbb{B}$-valued characteristic function $\chi_\nu : \mathbb{N} \to \mathbb{B}$ by $n \mapsto \{f \mid f(\nu, n) = 1\}$. This induces what $V^{\mathbb{B}}$ thinks is a new subset $\widetilde{\chi_\nu} \subseteq \mathbb{N}$, called a *Cohen real*, and furthermore, simultaneously performing this construction on all $\nu : \aleph_2$ induces what $V^{\mathbb{B}}$ thinks is a function from $\check{\aleph}_2 \to \mathcal{P}(\mathbb{N})$. After showing that $V^{\mathbb{B}}$ thinks this function is injective, to finish the proof it suffices to show that $x \mapsto \check{x}$ preserves cardinal inequalities, as then we will have squeezed $\check{\aleph}_1$ properly between $\mathbb{N}$ and $\mathcal{P}(\mathbb{N})$. This is really the technical heart of the matter, and centers on a combinatorial property of $\mathbb{B}$ called the *countable chain condition* (CCC), the proof of which requires a detailed combinatorial analysis of the basis of the product topology for $2^{\aleph_2 \times \mathbb{N}}$.

So far we have mentioned nothing about how this argument, which is wholly set-theoretic, is to be interpreted inside type theory. To do this, it was important for us to separate the mathematical content from the metamathematical content of the argument. Traditional presentations of forcing are careful to stay within the circle of ZFC, and it is not immediately clear what parts of the argument use that set-theoretic foundation in an essential way, e.g. if to perform forcing one must have ordinals that look like von Neumann ordinals in the metatheory, or if one must take a model of set theory as the starting point, or if $\mathbb{B}$ must be internal to a model of set theory, etc. As we will see, our formalization will clarify some of these questions.

Finally, we remark that for working with Boolean-valued models, it is profitable to keep in mind the following analogy, developed by Scott in [?]. A ready supply of complete Boolean algebras $\mathbb{B}$ is obtained by taking the measure algebra of a probability space and quotienting by the ideal of events of measure zero. Let $\mathbf{M}$ be a $\mathbb{B}$-valued structure. A unary $\mathbb{B}$-valued predicate ϕ on $\mathbf{M}$ assigns an event to every element m of $\mathbf{M}$, whose measure we can think of as being the probability that $\phi(m)$ is true. Specializing to the language of set theory, we can attach to every $m : \mathbf{M}$ an "indicator function" $\lambda x, x \in m$ which assigns to every x a probability that it is actually a member of m. Thus, by virtue of extensionality, we may think of the elements of a $\mathbb{B}$-valued model of ZFC as being "set-valued random variables", or "random sets".[3] We refer the interested reader to [?] and [?] for details.

Sources

Our strategy for constructing a Boolean-valued model in which the continuum hypothesis fails is a synthesis of the proofs in the textbooks of Bell ([?], Chapter 2) and Manin ([?], Chapter 8). For the Δ-system lemma, which is required for the countable chain condition, we followed Kunen ([?], Chapters 1 and 5).

Viewing the formalization

The code blocks in this paper should be taken as pseudocode. In some places, universe levels, type ascriptions, and casts have been removed to improve readability. We urge the interested reader to view our formalization.

The source code for our formalization is available at `https://github.com/flypitch/flypitch`. The forcing argument for the negation of CH is located in `forcing.lean`. In a Lean-aware editor such as Emacs, the user is encouraged start at the theorem `neg_CH` and jump backwards to trace the dependencies of the proof. (TODO(jesse) maybe change this to point at the "summary" file)

2 First-order logic

The starting point for first-order logic is a *language* of relation and function symbols. We represent a language as a pair of $\mathbb{N}$-indexed families of types, each of which is to be thought of as the collection of relation (resp. function) symbols, but stratified by arity.

[3] In this analogy, given a universe of random sets, the purpose of the generic filter or ultrafilter in forcing is then to simultaneously evaluate the outcomes of the random variables, collapsing them into an ordinary universe of sets.

```
structure Language : Type (u+1) :=
(functions : ℕ → Type u) (relations : ℕ → Type u)
```

2.1 (Pre)terms, (pre)formulas

TODO(floris) convert docstring into english

```
inductive preterm : ℕ → Type u
| var {} : ∀ (k : ℕ), preterm 0
| func : ∀ {l : ℕ} (f : L.functions l), preterm l
| app : ∀ {l : ℕ} (t : preterm (l + 1)) (s : preterm 0), preterm l
```

TODO(floris) convert docstring into english

```
inductive preformula : ℕ → Type u
| falsum {} : preformula 0
| equal (t₁ t₂ : term L) : preformula 0
| rel {l : ℕ} (R : L.relations l) : preformula l
| apprel {l : ℕ} (f : preformula (l + 1)) (t : term L) : preformula l
| imp (f₁ f₂ : preformula 0) : preformula 0
| all (f : preformula 0) : preformula 0
```

TODO(floris) convert docstring into english

```
inductive prf : set (formula L) → formula L → Type u
| axm     {Γ A} (h : A ∈ Γ) : prf Γ A
| impI    {Γ : set (formula L)} {A B} (h : prf (insert A Γ) B) : prf Γ
    (A ⟹ B)
| impE    {Γ} (A) {B} (h₁ : prf Γ (A ⟹ B)) (h₂ : prf Γ A) : prf Γ B
| falsumE {Γ : set (formula L)} {A} (h : prf (insert ∼A Γ) ⊥) : prf Γ
    A
| allI    {Γ A} (h : prf (lift_formula1 '' Γ) A) : prf Γ (∀' A)
| allE₂   {Γ} A t (h : prf Γ (∀' A)) : prf Γ (A[t // 0])
| ref     (Γ t) : prf Γ (t ≃ t)
| subst₂  {Γ} (s t f) (h₁ : prf Γ (s ≃ t)) (h₂ : prf Γ (f[s // 0])) :
    prf Γ (f[t // 0])
```

2.2 Completeness

As part of our formalization of first-order logic, we completed a verification of the Gödel completeness theorem. Although our present development of forcing did not require it, we anticipate that it will required later to e.g. prove the downward Löwenheim-Skolem theorem to extract countable transitive models for forcing with generic extensions; also, like the soundness theorem, it serves as a proof-of-concept and a stress-test of our chosen encoding of first-order logic.

For our formalization, we chose the Henkin-style approach of constructing a canonical term model. In order to perform the argument, which normally involves modifying the language “in place” to iteratively add new constant symbols, we had to adapt it to type theory. Since our languages are represented by pairs of indexed types instead of sets, we cannot really modify them in-place with new constant symbols. Instead, at each step of the construction, we must construct an entirely new language, in which the previous one embeds as a subtype, and in the limit, we cannot take a union, but must rather compute a directed colimit of types. This directed colimit of languages induces similar constructions on preterms, preformulas, sentences, and so on, and completing the argument requires reasoning with all of them. However, our proof was made significantly easier by our design decisions: only a few arguments required anything more than straightforward case-analysis and structural induction.

Here is one notable exception. In the process of our formalization, we discovered a nontrivial hole in many presentations of the Henkin argument (TODO(floris) describe `reflect_prf`)

We remark that our formalization of the completeness theorem is as general as possible, making no assumptions on the cardinality of the language or the presence of function symbols.

2.3 (Boolean-valued) soundness

2.4 Boolean-valued semantics for first-order logic

A **complete Boolean algebra** is a type $\mathbb{B}$ equipped with the structure of a Boolean algebra and additionally operations Inf and Sup (which we write as $\bigsqcap$ and $\bigsqcup$) returning the infimum and supremum of an arbitrary collection of members of $\mathbb{B}$. For more details on complete Boolean algebras, we refer the reader to the textbook of Halmos-Givant [?].

▸ **Definition 1.** *A Boolean-valued structure is...*

Note that Boolean-valued equality is not really an equivalence relation. One complication which then arises in Boolean-valued semantics is keeping track of the extensionality proofs for formulas. However, as part of the Soundness theorem shows, once these extensionality proofs are provided for the basic symbols in the language, they extend by structural induction to all formulas.

2.5 The soundness theorem

A soundness theorem says that a proof tree may be replayed to produce an actual proof in the object of truth-values. When the object is truth-values is the type `Prop` of propositions, this says that a proof tree compiles to a proof term. When the object of truth-values is a Boolean algebra, this says that the proof tree becomes an internal implication from the interpretation of the context to the interpretation of the conclusion.

We designed our datatype of proofs as an inductive type whose constructors are precisely the natural deduction rules naturally supported by Lean’s Prop. As a result, the proofs of either soundness theorem become straightforward structural inductions.

3 Constructing Boolean-valued models of set theory

Throughout this section, we fix a universe level u, a type $\mathbb{B}$ `: Type u` and an instance of a complete Boolean algebra structure on $\mathbb{B}$.

In set theory (see e.g. Jech [?] or Bell [?]), Boolean-valued models are obtained by imitating the construction of the von Neumann cumulative hierarchy via a transfinite recursion where iterations of the powerset operation (taking functions into $\mathbf{2} = \{\text{true}, \text{false}\}$) are replaced by iterations of the "$\mathbb{B}$-valued powerset operation" (taking functions into $\mathbb{B}$).

Since this construction by transfinite recursion does not easily translate into type theory, our construction of Boolean-valued models of set theory is instead a variation on a well-known encoding originally due to Aczel [?] [?] [?]. This encoding was adapted by Werner [?] to encode ZFC into Coq, whose metatheory is close to that of Lean. Werner's construction was re-implemented in Lean's `mathlib` by Carneiro as part of [?]. In this approach, one takes a universe of types `Type u` as the starting point and then imitates the cumulative hierarchy by constructing the inductive type

```
inductive pSet : Type (u+1)
| mk (α : Type u) (A : α → pSet) : pSet
```

(Just as the empty set kicks off the construction of the cumulative hierarchy, the empty type admits an empty map into any type and so induces `∅ : pSet`.)

The Aczel-Werner encoding is closely related to the recursive definition of *names*, which is used in forcing to construct forcing extensions:

▸ **Definition 2.** *Let P be a partial order (which one thinks of as a collection of forcing conditions). A* P-name *is a collection of pairs (y, p) where y is a P-name and $p : P$.*

If P consists of only one element, then a P-name is specified by essentially the same information as a member of the inductive type `pSet` above. Conversely, specializing P to an arbitrary complete Boolean algebra $\mathbb{B}$, we generalize the definition of `pSet.mk` so that elements are recursively assigned Boolean truth-values:

```
inductive bSet (𝔹 : Type u) [complete_boolean_algebra 𝔹] : Type (u+1)
| mk (α : Type u) (A : α → bSet) (B : α → 𝔹) : bSet
```

Thus `bSet 𝔹` is the type of $\mathbb{B}$-names, and will be the underlying type of our Boolean-valued model of set theory. For convenience, if `x : bSet 𝔹` and `x := ⟨α, A, B⟩`, we put `x.type := α, x.func := A, x.bval := B`.

3.1 Boolean-valued equality and membership

In `pSet`, equivalence of sets is defined by structural recursion as follows: two sets x and y are equivalent if and only if for every $w \in x$, there exists a $w' \in y$ such that w is equivalent to w', and vice-versa. Analogously, by translating quantifiers and connectives into operations on $\mathbb{B}$, Boolean-valued equality is defined in the same way:

```
def bv_eq : ∀ (x y : bSet 𝔹), 𝔹
| ⟨α, A, B⟩ ⟨α', A', B'⟩ :=
                  (⨅a : α, B a ⟹ ⨆a', B' a' ⊓ bv_eq (A a) (A' a')) ⊓
                    (⨅a' : α', B' a' ⟹ ⨆a, B a ⊓ bv_eq (A a) (A' a'))
```

We abbreviate `bv_eq` with the infix operator $=^{\mathbb{B}}$. With equality is place, it is easy to define membership, by translating "x is a member of y if and only if there exists a $w \in y$ such that $x = w$." As with equality, we denote $\mathbb{B}$-valued membership with $\in^{\mathbb{B}}$.

```
def mem : bSet 𝔹 → bSet 𝔹 → 𝔹
| a (mk α' A' B') := ⨆a', B' a' ⊓ a =ᴮ A' a'
```

3.2 Reasoning in 𝔹

As Scott stresses in [?], “A main point ... is that the well-known algebraic characterizations of [complete Heyting algebras] and [complete Boolean algebras] exactly mimic the rules of deduction in the respective logics…” Indeed, that is really why the Boolean-valued soundness theorem is true. One thinks of the $\leqslant$ symbol in an inequality of Boolean truth-values as a turnstile in a proof state: the conjunctands on the left as a list of assumptions in context, and the quantity on the right as the goal. For example, given `a b : 𝔹`, the identity $(a \Rightarrow b) \sqcap a \leqslant b$ could be proven by unfolding the definition of material implication, but it is really just the natural deduction rule of implication elimination; similarly, given an indexed family `a : I → 𝔹`, `⨆i, a i ≤ b ↔ ∀ i, a i ≤ b` is just casing on an existential quantifier.

Where the difficulty arises with having only a basic library of lemmas like the ones above is when the statements one wants to prove become not even nontrivial, but only slightly more complicated. Consider the following example, which should be “`by assumption`”:

```
 ∀ a b c d e f g: 𝔹, (d ⊓ e) ⊓ (f ⊓ g ⊓ ((b ⊓ a) ⊓ c)) ≤ a
```

or slightly less trivially, the following example where the goal is attainable by “just applying a hypothesis to an assumption”

```
 ∀ a b c d : 𝔹, (a ⟹ b) ⊓ c ⊓ (d ⊓ a) ≤ b
```

There are three ways to deal with goals like these, which approximately describe the evolution of our approach. First, one can try using the basic lemmas in `mathlib`, using the simplifier to normalize expressions, and performing clever rewrites with the deduction theorem[4]. Second, one can take the LCF-style approach and expand the library of lemmas with increasingly sophisticated derived inference rules.

Third, one can make the following observation:

▸ **Lemma 3.** *Let $(P, \leqslant)$ be a partially ordered set. Let $a\, b : P$. Then $a \leqslant b$ if and only if $\forall \Gamma : P, \Gamma \leqslant a \to \Gamma \leqslant b$.*

This is an instance of the Yoneda lemma for partially ordered sets, and its proof is utterly trivial. However, one side of the equivalence is much easier for Lean to reason with. Take the example which should have been “`by assumption`”. The following proof, in which the user navigates down the binary tree of nested $\sqcap$s, will work:

```
example {a b c d e f g : 𝔹} : (d ⊓ e) ⊓ (f ⊓ g ⊓((b ⊓ a)⊓ c)) ≤ a :=
by {apply inf_le_right_of_le, apply inf_le_right_of_le,
    apply inf_le_left_of_le, apply inf_le_right_of_le, refl}
```

But if we use the right-hand side of Lemma 3 instead, then after some preprocessing, `assumption` will literally work:

[4] The deduction theorem in a Boolean algebra says that for all a, b and c, $a \sqcap b \leqslant c \iff a \leqslant b \Rightarrow c$.

```
example {a b c d e f g : 𝔹} : (d ⊓ e) ⊓ (f ⊓ g ⊓((b ⊓ a)⊓ c)) ≤ a :=
by {apply poset_yoneda, intros Γ H, simp only [le_inf_iff] at H,
  repeat{auto_cases}, assumption}
/- Goal state before `assumption`:
[...]
H_right_left_right : Γ ≤ g,
H_right_right_left_left : Γ ≤ b,
H_right_right_left_right : Γ ≤ a
⊢ Γ ≤ a -/
```

Automation and metaprogramming

A key feature of Lean is that it is its own metalanguage, allowing for seamless in-line definitions of custom tactics. This feature was an invaluable asset, as it allowed the rapid development of a custom tactic library for simulating natural-deduction style proofs inside 𝔹 after applying Lemma 3 to insert a slack context variable Γ. The preprocessing steps before the call to `assumption` in the previous example are bundled into a single tactic `tidy_context`, and Boolean-valued versions of basic niceties like or-elimination, instantiation of existentials, implication introduction, and even basic automation were easy to write and considerably streamlines the formalization workflow, to the point where the user is able to pretend, with absolute rigor, that they are simply writing proofs in first-order logic while calculations in the complete Boolean algebra are being performed under the hood. **TODO(jesse) wording**

One use-case where automation is crucial is context-specialization ("change of variables"). For example, if, after preprocessing with `poset_yoneda`, the goal is Γ ≤ `a` ⟹ `b`, and one would like to "introduce the implication", by adding Γ ≤ `a` to context and reducing the goal to Γ ≤ `b`, this is impossible as stated. Rather, the deduction theorem lets us rewrite the goal to Γ ⊓ `a` ≤ `b`, and now we may add Γ ⊓ `a` ≤ `a`. So we may introduce the implication after all, but at the cost of specializing the context Γ to the smaller context Γ′ `:=` Γ ⊓ `a`. But now, in order for the user to continue the pretense that they are merely doing first-order logic, this change of variables must be propagated to the rest of the assumptions which may still be of the form Γ ≤ `_`—which is extremely tedious to do by hand, but easy to automate.

(In a sense, these tactics are only a substitute for a yet-to-be implemented framework for proof by reflection using the Boolean-valued soundness theorem... however, there are important cases which are inaccessible by just re-interpreting proofs of ZF... TODO(jesse) maybe delete)

3.3 Check-names

From the definitions of `pSet` and `bSet`, one immediately sees that there is a canonical map `check : pSet → bSet 𝔹`, defined by

```
def check : pSet → bSet 𝔹
| ⟨α,A⟩ := ⟨α, λ a, check (A a), λ a, ⊤⟩
```

That is, `check` takes a `pSet` and recursively attaches the Boolean truth-value ⊤ to all elements. We call members of the image of `check` *check-names*. These are also known as *canonical names*, as they are the canonical representation of standard two-valued sets inside a Boolean-valued model of set theory.

One of the most important considerations in forcing is how cardinals and ordinals in the metatheory interact with cardinals and ordinals in the forcing extension. We will see later that after translating the entire forcing argument to type theory, the presence of `pSet` is misleading. It is not the ordinals and cardinals inside `pSet` which are of fundamental importance, but rather the ordinals and cardinals of Lean itself. The role of the check-names is not any less decisive, but is it clear that the check-names are merely the way in which the metatheory's cardinals and ordinals interact with that in the forcing extension. `pSet` is not a prerequisite for studying `bSet` or performing forcing, but only a convenient aid to organize information about the check-names.

3.4 Transfinite induction

In set theory, it is common to prove propositions via induction on an ordinal-valued rank function. In fact, this is how V^{BB} is typically constructed, by induction on the rank of sets in an existing universe of sets V. In Lean, this style of argument does not come for as free as, say, structural induction principles like ∈-induction, which by virtue of the construction of bSet BB, *is* the induction principle for that inductive type. However, an interface is available for well-founded recursion on well-founded relations, and a development of the theory of ordinals as equivalence-classes of well-ordered types is available in `mathlib`. There were two places in the present work where transfinite induction was unavoidable, namely in the construction of an antichain for the maximum principle, and the verification that the canonical embedding of ordinals into `pSet` is injective.

3.5 The fundamental theorem of forcing

The fundamental theorem of forcing for Boolean-valued models [?] states that for any complete Boolean algebra B, V^B is a Boolean-valued model of ZFC. Since, in type theory, a type universe `Type u` takes the place of the standard universe V, the analogous statement in our setting is that for every complete Boolean algebra $\mathbb{B}$, `bSet 𝔹` is a Boolean-valued model of ZFC.

After the development of the custom proof language, **(TODO(jesse) add more definitions to ref to** much of the verification of the axioms besides choice is routine, as the user is able to pretend they are working in ordinary 2-valued logic. We describe some aspects of `bSet 𝔹` which are illuminated by the verification of the axioms and which will be important for forcing ¬CH.

The axiom of infinity

ω `: bSet 𝔹` is $\check{\omega}$. ω is defined in `pSet` to be the collection of all finite von Neumann ordinals, which are defined by induction on ℕ. While it is easy to show $\check{\omega}$ satisfies the axiom of infinity

```
def axiom_of_infinity_spec (u : bSet 𝔹) : 𝔹 :=
  (∅∈ᴮ u) ⊓ (⨅i_x, ⨆i_y, (u.func i_x ∈ᴮ u.func i_y))
```

it can furthermore be shown to satisfy the universal property of ω, which says that ω is a subset of any set which contains $\varnothing$ and is closed under the successor operation $x \mapsto x \cup x$.

The axiom of powerset

▸ **Definition 4.** *Fix a* $\mathbb{B}$*-valued set* $x = \langle \alpha, A, b \rangle$. *Let* $\chi : \alpha \to \mathbb{B}$ *be a function. The subset of* x *associated to* χ *is a* $\mathbb{B}$*-valued set defined as follows:*

```
def set_of_indicator {x} (χ : x.type → 𝔹) := ⟨x.type, x.func, χ⟩
```

*The **powerset** $\mathcal{P}(x)$ of x is defined to be the following $\mathbb{B}$-valued set, whose underlying type is the type of all functions* `x.type` $\to$ $\mathbb{B}$*:*

```
def bv_powerset (u : bSet 𝔹) : bSet 𝔹 :=
⟨u.type → 𝔹, λ f, set_of_indicator f, λ f, set_of_indicator f ⊆ᴮ u⟩
```

The axiom of choice

Following the presentation in Bell [?], we verified Zorn's lemma, which is provably equivalent over ZF to the axiom of choice. As is the case with `pSet`, establishing the axiom of choice requires the use of a choice principle from the metatheory. This was the most involved part of our verification of the fundamental theorem of forcing, and relies on the technical tool of *mixtures*, which allow sequences of $\mathbb{B}$-valued sets to be "averaged" into new ones, and the *maximum principle*, which allows existentially quantified statements to be instantiated without changing their truth-value.

The smallness of $\mathbb{B}$

Before ending this section, we remark that the "smallness" (or more precisely, the fact that $\mathbb{B}$ lives in the same universe of types out of which `bSet 𝔹` is being built), plays a crucial a role in making `bSet 𝔹` a model of ZFC. It is required for extracting the witness needed for the maximum principle, and is also required to even define the powerset operation, because the underlying type of the powerset is the function type of all maps into $\mathbb{B}$.

4 Forcing

4.1 Representing Lean's ordinals inside `pSet` and `bSet`

The treatment of ordinals in `mathlib` associates a class of ordinals to every type universe, defined as isomorphism classes of well-ordered types, and includes interfaces for both well-founded and transfinite recursion. Lean's ordinals may be represented inside `pSet` by defining a map `ordinal.mk : ordinal → pSet` via transfinite recursion; it is nothing more than the von Neumann definition of ordinals. In pseudocode,

```
def ordinal.mk : ordinal → pSet
| 0 := ∅
| succ ξ := pSet.succ (ordinal.mk ξ) -- (mk ξ ∪ {mk ξ})
| is_limit ξ := ⋃ η < ξ, (ordinal.mk η)
```

Composing by `check` (??) yields a map `check ∘ ordinal.mk : ordinal → bSet 𝔹`. (We could just as well defined `ordinal.mk′ : ordinal → bSet 𝔹` analogously to `ordinal.mk` such that `ordinal.mk′ = check ∘ ordinal.mk`; the point is that there is a link between the metatheory's notion of size and order with that of the forcing extension.)

Cardinals are defined separately from ordinals as bijective equivalence classes of types, but are canonically represented by ordinals which is are not bijective with any predecessor. We let `aleph : ordinal → ordinal` index these representatives. For the rest of this section, unadorned alephs (e.g. "$\aleph_2$") will mean either an ordinal of the form `aleph` ξ or a choice of representative from the isomorphism class of well-ordered types, and checked alephs (e.g. "$\check{\aleph_2}$") will mean the `check ∘ ordinal.mk` of that ordinal.

4.2 The Cohen poset and the regular open algebra

Forcing with partial orders and forcing with complete Boolean algebras are related by the fact that every poset of forcing conditions can be embedded into a complete Boolean algebra as a dense suborder. This will be the case for our forcing argument: our Boolean algebra is the algebra of regular opens on $2^{\aleph_2 \times \mathbb{N}}$, which embeds the poset of forcing conditions typically used for Cohen forcing as a dense suborder.

▸ **Definition 5.** *The **Cohen poset** for adding $\aleph_2$-many Cohen reals is the collection of all finite partial functions $\aleph_2 \times \mathbb{N} \to \mathbf{2}$, ordered by reverse inclusion.*

In the formalization, the Cohen poset is represented as a `structure` with three fields:

```
structure 𝓒 : Type :=
  (ins : finset (ℵ₂.type × ℕ))
  (out : finset (ℵ₂.type × ℕ))
  (H : ins ∩ out = ∅)
```

That is, we identify a finite partial function f with the triple ⟨`f.ins`, `f.out`, `f.H`⟩, where `f.ins` is the preimage of $\{1\}$, `f.out` is the preimage of $\{0\}$, and `f.H` ensures well-definedness. While f is usually defined as a finite partial function, we found that in practice f is really only needed to give a finite partial specification of a subset of $\aleph_2 \times \mathbb{N}$ (i.e. a finite set `f.ins` which *must* be in the subset, and a finite set `f.out` which *must not* be in the subset), and chose this representation to make that information immediately accessible.

▸ **Definition 6.** *Let X be a topological space, and for any open set U, let $U^\perp$ denote the complement of the closure of U. The **regular open algebra** of a topological space X, written* $\mathrm{RO}(X)$*, is the collection of all open sets U such that $U = (U^\perp)^\perp$, equipped with the structure of a complete Boolean algebra, with $x \sqcap y := x \cap y$, $x \sqcup y := ((x \sqcup y)^\perp)^\perp$, $\neg x := x^\perp$, and* $\bigsqcup x_i := ((\bigcup x_i)^\perp)^\perp$.

The Boolean algebra which we will use for forcing ¬CH is $\mathrm{RO}(2^{\aleph_2 \times \mathbb{N}})$. Unless stated otherwise, for the rest of this section, we put $\mathbb{B} := \mathrm{RO}(2^{\aleph_2 \times \mathbb{N}})$.

▸ **Definition 7.** *We define the **canonical embedding** of the Cohen poset into $\mathbb{B}$ as follows:*

```
def ι : 𝓒 → 𝔹 := λ p, {S | p.ins ⊆ S ∧ p.out ⊆ - S}
```

That is, we send each `c : 𝓒` all the subsets which satisfy the specification given by `c`. This is a clopen set, hence regular. Crucially, this embedding is *dense*:

```
lemma 𝓒_dense {b : 𝔹} (H : ⊥ < b) : ∃ p : 𝓒, ι p ≤ b
```

Recalling that $\leqslant$ in $\mathbb{B}$ is subset-inclusion, we see that this is essentially because the image of $\iota : \mathcal{C} \to \mathbb{B}$ *is* is the standard basis for the product topology. Our chosen encoding of the Cohen poset also made it easier to perform this identification when formalizing this proof.

4.3 Adding $\aleph_2$-many distinct Cohen reals

As we saw in ??, for any $\mathbb{B}$-valued set x, characteristic functions into $\mathbb{B}$ from the underlying type of x determine $\mathbb{B}$-valued subsets of x. While the ingredients $\aleph_2$ and $\mathbb{N}$ for $\mathbb{B}$ are types and thus external to `bSet 𝔹`, they are represented nonetheless inside `bSet 𝔹` by their check-names $\check{\aleph}_2$ and $\check{\mathbb{N}}$, and in fact $\aleph_2$ *is* $\aleph_2\check{}$ `.type` and $\mathbb{N}$ *is* $\check{\mathbb{N}}$`.type`. Given our specific choice of $\mathbb{B}$, this will allow us to construct an $\aleph_2$-indexed family of distinct subsets of $\check{\mathbb{N}}$, which we can then convert into an injective function from $\aleph_2\check{}$ to $\mathbb{N}$, *inside* `bSet 𝔹`.

▸ **Definition 8.** *Let $\nu : \aleph_2$. For any $n : \mathbb{N}$, the collection of all subsets of $\aleph_2 \times \mathbb{N}$ which contain (ν, n) is a regular open of $2^{\aleph_2 \times \mathbb{N}}$, called the* ***principal open*** $\mathbf{P}_{(\nu,n)}$ *over (ν, n).*

▸ **Definition 9.** *Let $\nu : \aleph_2$. We associate to ν the $\mathbb{B}$-valued characteristic function $\chi_\nu : \mathbb{N} \to \mathbb{B}$ defined by $\chi_\nu(n) := \mathbf{P}_{(\nu,n)}$. In light of our previous observations, we see that each χ_ν induces a new $\mathbb{B}$-valued subset $\widetilde{\chi_\nu} \subseteq \check{\mathbb{N}}$. We call $\widetilde{\chi_\nu}$ a* ***Cohen real****.*

This gives us an $\aleph_2$-indexed family of Cohen reals. Converting this data into an injective function from $\check{\aleph_2}$ to $\mathbb{N}$ inside `bSet` $\mathbb{B}$ requires some care. One must check that $\nu \mapsto \widetilde{\chi_\nu}$ is externally injective, and this is where the characterization of the Cohen poset as a dense subset of $\mathbb{B}$ (and moving back and forth between this representation and the definition as finite partial functions) comes in. Furthermore, one has to develop machinery similar to that for the powerset operation to convert an external injective function `x.type` $\to$ `bSet` $\mathbb{B}$ to a $\mathbb{B}$-valued set which `bSet` $\mathbb{B}$ believes is a injective function, while maintaining conditions on the intended codomain. Our custom tactics and automation for reasoning inside $\mathbb{B}$ made this latter task significantly easier than it would have been otherwise. We refer the interested reader to our formalization for details.

4.4 Preservation of cardinal inequalities

So far, we have shown that for $\mathbb{B} = \mathrm{RO}(2^{\aleph_2 \times \mathbb{N}})$, `bSet` $\mathbb{B}$ thinks $\check{\aleph_2}$ is smaller than $\mathcal{P}(\check{\mathbb{N}})$.

Although Lean believes there is a strict inequality of cardinals $\aleph_0 < \aleph_1 < \aleph_2$, in general we can only deduce that their representations inside `bSet` $\mathbb{B}$ are subsets of each other: $\top \leqslant \check{\aleph_0} \subseteq^{\mathbb{B}} \check{\aleph_1} \subseteq^{\mathbb{B}} \check{\aleph_2}$. To finish negating CH, it suffices to show that `bSet` $\mathbb{B}$ believes $\check{\aleph_0}$ is strictly smaller than $\check{\aleph_1}$, and that `bSet` $\mathbb{B}$ believes $\check{\aleph_1}$ is a strictly smaller than $\check{\aleph_2}$. That is, we want that the passage from $\aleph_i$ to $\check{\aleph_i}$ preserves cardinal inequalities.

▸ **Definition 10.** *For our purposes, "strictly smaller" will mean "there exists no function f such that for every `v` $\in$ `y`, there exists a `w` $\in$ `x` such that `(w,v)` $\in$ `f`". Translated to a Boolean truth-value (with the definition of a function abbreviated), this means: the Boolean truth-value of "`x` is strictly smaller than `y`" is defined to be*

$$-(\bigsqcup f,\ (\mathit{is\_func}\ f) \sqcap \bigsqcap v,\ v \in^{B} y \implies \bigsqcup w,\ w \in^{B} x \sqcap \mathit{pair}\ w\ v \in^{B} f)$$

The condition on an arbitrary $\mathbb{B}$ which ensures the preservation of cardinal inequalities is the *countable chain condition.*

▸ **Definition 11.** *We say that $\mathbb{B}$ has the* ***countable chain condition*** *(CCC) if every antichain $\mathcal{A} : I \to \mathbb{B}$ (i.e. an indexed collection of elements $\mathcal{A} := \{a_i\}$ such that whenever $i \neq j, a_i \sqcap a_j = \bot$) has a countable image.*

We sketch the argument that CCC implies the preservation of cardinal inequalities. The proof is by contraposition. Let κ_1 and κ_2 be cardinals such that $\kappa_1 < \kappa_2$, and suppose that $\check{\kappa_1}$ is not strictly smaller than $\check{\kappa_2}$. Then there exists some `f : bSet` $\mathbb{B}$ and some $\Gamma > \bot$ such that $\Gamma \leqslant$ `(is_func f)` $\sqcap \bigsqcap$`v, v` $\in^B \check{\kappa_1} \implies \bigsqcup$`w, w` $\in^B \check{\kappa_2} \sqcap$ `(w,v)` $\in^B$ `f`. Then one can show:

```
lemma AE_of_check_larger_than_check :
∀ β < κ₂, ∃ η < κ₁, ⊥ < (is_func f) ⊓ (η̌, β̌ ) ∈ᴮ f
```

The name of this lemma emphasizes that what was happened here is that, given this f and the assumption that it satisfes some $\forall$-$\exists$ formula inside `bSet` $\mathbb{B}$, we are able to extract, by virtue of $\check{\kappa_1}$ and $\check{\kappa_2}$ being check-names, a $\forall$-$\exists$ statement in the *metatheory.* Using Lean's choice principle, we can then convert this ∀-∃ statement into a function $g : \kappa_2 \to \kappa_1$, such that for every β, `⊥ < (is_func f) ⊓ (g(β)ˇ , β̌ ) ∈ᴮ f`. Since $\kappa_2 > \kappa_1$, it follows from the infinite pigeonhole principle that there exists some $\eta < \kappa_1$ such that the $g^{-1}(\{\eta\})$ is uncountable. Define $\mathcal{A} : g^{-1}(\{\eta\}) \to \mathbb{B}$ by $\mathcal{A}(\beta) :=$ `(is_func f) ⊓ (g(β)ˇ , β̌ ) ∈ᴮ f`. This is an uncountable antichain because if $\beta_1 \neq \beta_2$, then the well-definedness part of `is_func f` ensures that, because $g(\beta_1) = g(\beta_2)$, the truth-value $\check{\beta_1} = f(g(\beta_1)) \neq^{\mathbb{B}} f(g(\beta_2)) = \check{\beta_2}$ is $\bot$.

Thus, conditional on showing that $\mathbb{B} = \mathrm{RO}(2^{\aleph_2 \times \mathbb{N}})$ has the CCC, we now have that cardinal inequalities are preserved in `bSet` $\mathbb{B}$. Combining this with the injection $\check{\aleph_2} \to \mathcal{P}$ $(\mathbb{N})$, we obtain:

```
theorem neg_CH : ⊤ ≤ ℕ ≺ (ℵ₁)ˇ ≺ (ℵ₂)ˇ ≼ 𝒫(ℕ)
```

The arguments sketched in subsection 4.3 and **??** form the heart of the forcing argument. Their proofs involve taking objects in `Type u` and `bSet` $\mathbb{B}$, constructing corresponding objects on the other side, and reasoning about them in ordinary and $\mathbb{B}$-valued logic simultaneously to determine cardinalities in `bSet` $\mathbb{B}$. We have omitted many details from our discussion, but of course, all details of the proofs have been formally verified.

5 Transfinite combinatorics and the countable chain condition

What remains now is to prove that $\mathrm{RO}(2^{\aleph_2 \times \mathbb{N}})$ has the CCC. There are several ways forward, but we chose the most general, anticipating its usefulness in future formalizations of set theory and forcing.

TODO(floris)

5.1 The Δ-system lemma

TODO(floris)

5.2 $\mathrm{RO}(2^{\aleph_2 \times \mathbb{N}})$ has the countable chain condition

TODO(floris)

6 Conclusions and future work

6.1 Proof by reflection

Combined with the usual completeness theorem, the Boolean-valued soundness theorem will allow us to prove statements about a structure of the form bSet BB as follows: if the statement ϕ is provable from ZF, then we may prove that ZF proves ϕ by applying the completeness theorem to reason inside an arbitrary model of ZF. This avoids the complications of trying to work directly inside Boolean-valued logic. Then, given a Boolean-valued $\mathrm{L}_{\mathrm{ZFC}}$-structure $\mathbf{M}$ which satisfies ZF, the Boolean-valued soundness theorem tells us that this proof may be replayed inside $\mathbf{M}$ so that ϕ has truth-value greater than the truth-values of ZF, and is therefore satisfied inside $\mathbf{M}$.

In this way, we can transport proofs of statements such as "Zorn's lemma is equivalent to the axiom of choice", which is provable from ZF, directly from the world of 2-valued models to the world of Boolean-valued models.

We also remark that while our use of `simp` lemmas to generate congruence certificates sufficed for the purposes of this work, the "real" proof that something like the subset predicate is $=^B$-extensional is a proof by reflection: one constructs a formula which reifies the predicate, and then applies the fact that one is in the deeply-embedded Boolean-valued structure to obtain the congruence lemma automatically. We also intend to automate this.

6.2 Forcing with generic models

Our method does not support iterated forcing. Our method starts with a universe of types and uses that to construct a model of set theory.

6.3 Towards a formal proof of the independence of the continuum hypothesis

This work was carried out as part of the Flypitch project, which aims to formalize the independence of the continuum hypothesis from ZFC, i.e. that CH and its negation are both unprovable from the ZFC axioms.

Future goals of the project include:

- Various formulations of the axioms of ZFC are equiconsistent, including the versions used in this paper

- In order to complete this formalization, we had to develop several libraries, e.g. for dependently-typed vectors, significant extensions of the lattice and Boolean algebra library, the theory of product topological space and their bases, and extensions to the set theory and ordinal libraries.
- `pSet` was not essential. Rather, in our type-theoretic foundations, the construction of a Boolean-valued standard universe of set theory has equal footing with the construction of an ordinary standard universe of set theory. We see that for the purposes of working with $V^{\mathbb{B}}$, V is no longer a prerequisite, but merely a useful tool for organizing the check-names.
- We used several features of our type-theoretic foundations to our advantage. We constructed a standard universe of set theory structurally in such a way that many properties of the underlying universe of types are reflected inside the model of set theory, and such that we get the axiom of regularity (more precisely, the principle of epsilon-induction) for free as the automatically-generated induction principle for our inductive type.
- Lean is great, meteoric growth — remark on recentness of developments in `mathlib` which made this possible (acknowledge developments from other theorem-provers, including porting of libraries e.g. `lattice` from Isabelle).
- Evidence that formalized mathematics is ready "in the large"

7 References

- Moore's The method of forcing
- Halmos-Givant Textbook on boolean algebras
- Gunther Pagano et al forcing in Isabelle/ZF
- Paulson constructible universe and set theory in Isabelle/ZF
- Sets in Coq, Coq in Sets
- Sets in types, types in sets
- Aczel's encoding of ZFC inside type theory