A Formal Proof of the Independence of the Continuum Hypothesis

Jesse Michael Han
Department of Mathematics
University of Pittsburgh
Pittsburgh, PA, USA
jessemichaelhan@gmail.com

Floris van Doorn
Department of Mathematics
University of Pittsburgh
Pittsburgh, PA, USA
fpvdoorn@gmail.com

Abstract

We describe a formal proof of the independence of the continuum hypothesis (CH) in the Lean theorem prover. We use Boolean-valued models to give forcing arguments for both directions, using Cohen forcing for the consistency of ¬CH and a σ-closed forcing for the consistency of CH.

• **Theory of computation → Logic and verification**; **Type theory**.

Interactive theorem proving, formal verification, continuum hypothesis, forcing, Lean, set theory, ZFC, Boolean-valued models

ACM Reference Format:
Jesse Michael Han and Floris van Doorn. 2020. 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

1 Introduction

The continuum hypothesis (CH) states that there is no cardinality between ω, the smallest infinite cardinal and $\mathfrak{c}$, the cardinality of the continuum. It was posed by Cantor [6] in 1878 and was the first problem on Hilbert's list of twenty-three unsolved problems in mathematics. Gödel [14] proved in 1938 that CH was consistent with Zermelo-Fraenkel set theory with the axiom of choice (ZFC). He conjectured that CH was independent, i.e. neither provable nor disprovable, from ZFC. This remained an open problem until 1963, when Paul Cohen developed *forcing* [8, 9] and used it to prove the consistency of ¬CH with ZFC, completing the independence proof. This work started modern set theory, and for his invention of forcing, Cohen was awarded a Fields medal.

The independence of CH has also been an open formalization problem. Since 2005, Freek Wiedijk has maintained a list (*Formalizing 100 theorems* [47]) of one hundred problems for formalized mathematics, with the independence of CH as the 24th. As of 2019, it was one of the six remaining problems.

In this paper we describe the successful completion of the Flypitch project[1] (**F**orma**l**l**y** **p**roving the **i**ndependence of **t**he **c**ontinuum **h**ypothesis). We formalize forcing with Boolean-valued models. We use Cohen forcing to construct a Boolean-valued model of ZFC where CH is false, and a σ-closed forcing to construct a Boolean-valued model of ZFC where CH is true. We then combine this with a deep embedding of first-order logic, including a proof system and the axioms of ZFC, to verify that CH is neither provable nor disprovable from ZFC.

Our formalization[2] uses the Lean 3 theorem prover, building on top of mathlib [29]. Lean is an interactive proof assistant under active development at Microsoft Research [10, 44]. It has a similar metatheory to Coq, adding definitional proof irrelevance, quotient types, and a noncomputable choice principle. 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, let alone ZF. The types of cardinals and ordinals in mathlib, which are defined as proper equivalence classes of (well-ordered) types, live one universe level higher than the types used to construct them, and our models of set theory require as input an entire universe of types. Our encoding of first-order logic also uses parameterized inductive types which ensure that type-correctness implies well-formedness, eliminating the need for separate well-formedness proofs.

The method of forcing with Boolean-valued models was developed by Solovay and Scott [38, 40] as a simplification of Cohen's method. Some of these simplifications were incorporated by Shoenfield [43] into a general theory of forcing using partial orders, and it is in this form that forcing is usually

CPP '20, January 20–21, 2020, New Orleans, LA, USA

ACM ISBN 978-1-4503-7097-4/20/01…$15.00
https://doi.org/10.1145/3372885.3373826

[1] https://flypitch.github.io
[2] https://github.com/flypitch/flypitch

practiced. While both approaches have essentially the same mathematical content (see e.g. [25, 26, 30]), there are several reasons why we chose to use Boolean-valued models. The main reason is the directness of forcing with Boolean-valued models, which bypasses the need for the Löwenheim-Skolem theorems, Mostowski collapse, countable transitive models, or genericity considerations for filters. The theory of forcing with Boolean-valued models also cleanly splits into several parts, allowing us formalize different components in parallel (e.g. a general theory of Boolean-valued semantics, a library for calculations in complete Boolean algebras, a construction of Boolean-valued models of set-theory) and later recombine them. In particular, our library for Boolean-valued semantics for first-order logic is completely general and can be reused for other formalization projects. Finally, our Boolean-valued models of set theory are inductive types generalizing the Aczel encoding of set theory into dependent type theory; consequently, the automatically-generated induction principle *is* $\in$-induction, leading to cleaner proofs.

1.1 Proof Outline

The usual method to show that a statement is unprovable is to construct a model where the statement is false, and apply the soundness theorem; our method is similar, except that we use Boolean-valued semantics and a Boolean-valued soundness theorem (see Section 3). The difference between Boolean-valued models and ordinary models is that the truth values in a Boolean-valued model `M` live in a complete Boolean algebra $(\mathbb{B}, \sqcap, \sqcup, \bigsqcap, \bigsqcup, \bot, \top)$. If we can construct two Boolean-valued models of ZFC, one where CH is true $\top$, and one where CH is false $\bot$, then by the Boolean-valued soundness theorem, CH is independent from ZFC.

For any complete Boolean algebra $\mathbb{B}$ we implement the set-theoretic universe $V^{\mathbb{B}}$ of $\mathbb{B}$-valued sets by generalizing the Aczel encoding of set theory (called `pSet`, see Section 4), obtaining a type `bSet` $\mathbb{B}$ of $\mathbb{B}$-valued sets. The fundamental theorem of forcing for Boolean-valued models [17], translated to our situation, then states that `bSet` $\mathbb{B}$ is a $\mathbb{B}$-valued model is ZFC.

To show the independence of CH, it remains to construct two appropriate complete Boolean algebras The properties of `bSet` $\mathbb{B}$ can vary wildly depending on the choice of the complete Boolean algebra $\mathbb{B}$. There is always a map `check` : `pSet` $\to$ `bSet` $\mathbb{B}$, $x \mapsto \check{x}$, but in general, $\check{x}$ might have different properties than x. Making a good choice of $\mathbb{B}$ and controlling the behavior of the check-names is precisely the task of forcing (Section 5).

Traditional presentations of forcing, even with Boolean-valued models (e.g. [4], [25]), are careful to stay within the foundations of ZFC, emphasizing that all arguments may be performed internal to a model of ZFC, etc. In order to formalize these set-theoretic arguments in a type-theoretic metatheory, it is important to separate their mathematical content from their metamathematical content. It is not immediately clear what parts of these arguments use their set-theoretic foundation in an essential way and require modification in the passage to type theory. Our formalization clarifies some of these questions.

We use custom domain-specific tactics and various forms of automation throughout our formalization, notably a tactic library for simulating natural deduction proofs inside a complete Boolean algebra Section 6). This reveals another advantage of working in a proof assistant: the bookkeeping of Boolean truth-values, sometimes regarded as a tedious aspect of the Boolean-valued approach to forcing, can be automated away.

Contributions An earlier paper [18] describes a formalization of Cohen forcing and the unprovability of CH. In order to keep our presentation self-contained, we reproduce some of that material here, incorporating it into our discussions of our deep embedding of first-order logic/Boolean-valued semantics, usage of metaprogramming, and the Cohen forcing argument. Our main novel contribution is a formalization of collapse forcing and the unprovability of ¬CH, thereby providing the first formalization of the independence of CH in a single theorem prover. For reasons we will see in Section 5, the forcing argument for CH requires far more set theory and is harder to formalize than the forcing argument for ¬CH. Moreover, we elaborate on parts of the formalization which were omitted from [18], including expanded discussions of our implementation of the ZFC axioms and our formalization of the Δ-system lemma.

Sources Our strategy for forcing ¬CH is a synthesis of the proofs in the textbooks of Bell ([4], Chapter 2) and Manin ([27], Chapter 8). For the Δ-system lemma, which we use to verify that Cohen forcing is CCC, we follow Kunen ([26], Chapters 1 and 5).

We were unable to find a reference for a purely Boolean-valued account of forcing CH. We loosely followed the conventional arguments given by Weaver ([45], Chapter 12) and Moore ([30]), and base our construction of $\mathbb{B}_{\text{collapse}}$ on the collapse algebras defined by Bell ([4], Exercise 2.18).

Related Work Set theory and first-order logic are both common targets for formalization. Shankar [41] used a deep embedding of first-order logic for incompleteness theorems. Harrison gives a deeply-embedded implementation of first-order logic in HOL Light [19] and a proof-search style account of the completeness theorem in [20]. Other formalizations of first-order logic can be found in Isabelle/HOL ([36], [37],[5]) and Coq ([24], [31]).

A large body of formalized set theory has been completed in Isabelle/ZF, led by Paulson and his collaborators [32, 33, 35], including the relative consistency of AC with ZF [34]. Building on this, Gunther, Pagano, and Terraf have taken

some first steps towards formalizing forcing [15, 16], by way of generic extensions of countable transitive models.

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 stratified by arity:

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

2.1 Terms, Formulas and Proofs

The main novelty of our implementation of first-order logic is the use of *partially applied* terms and formulas, encoded in a parameterized inductive type where the $\mathbb{N}$ parameter measures the difference between the arity and the number of applications. The benefit of this is that it is impossible to produce an ill-formed term or formula, because type-correctness is equivalent to well-formedness. This eliminates the need for separate well-formedness proofs.

Fix a language L. We define the type of **preterms** as follows:

```
inductive preterm (L : Language.{u}) :
    ℕ → Type u
| var : ℕ → preterm 0 -- notation `&`
| func {l : ℕ} : L.functions l → preterm l
| app {l : ℕ} :
    preterm (l + 1) → preterm 0 → preterm l
```

A member of `preterm n` is a partially applied term. If applied to `n` terms, it becomes a term. We define the type of well-formed terms `term L` to be `preterm L 0`.

The type of **preformulas** is defined similarly:

```
inductive preformula (L : Language.{u}) :
    ℕ → Type u
| falsum : preformula 0 -- notation ⊥
| equal : term L → term L → preformula 0
    -- notation ≃
| rel {l : ℕ}, L.relations l → preformula l
| apprel {l : ℕ}, preformula (l + 1) →
    term L → preformula l
| imp : preformula 0 → preformula 0 →
    preformula 0 -- notation ⟹
| all : preformula 0 → preformula 0
    -- notation ∀'
```

We choose this definition of `preformula` to mimic `preterm`. A member of `preformula n` is a partially applied formula, and if applied to `n` terms, it becomes a formula. The type of well-formed formulas `formula L` is defined to be `preformula L 0`. Implication is the only primitive binary connective and universal quantification is the only primitive quantifier. Since we use classical logic, we can define the other connectives and quantifiers from these. Note that implication and the universal quantifier cannot be applied to preformulas that are not fully applied.

It is also possible to define well-typed terms and formulas using vectors of terms and nested inductive types. However, we avoided these kinds of definitions because Lean has limited support for nested inductive types. In the case of formulas, this would not even result in a nested inductive type, but we found it more convenient to adapt operations and proofs from `preterm` to `preformula` using our definition.

We use de Bruijn indices to avoid variable shadowing. This means that the variable `&m` under `k` is bound if $m < k$ and otherwise represents the $(m-k)$-th free variable. We define the usual operations of lifting and substitution for terms and formulas, needed when using de Bruijn variables. The notation `t ↑' n # m` means the preterm of preformula `t` where all variables which are at least `m` are increased by `n`. The lift `t ↑' n # 0` is abbreviated to `t ↑ n`. The substitution `t[s // n]` is defined to be the term or formula `t` where all variables that represent the `n`-th free variable are replaced by `s`. More specifically, if an occurrence of a variable `&(n+k)` is under `k` quantifiers, then it is replaced by `s ↑ (n+k)`. Variables `&m` for $m > n + k$ are replaced by `&(m-1)`.

Our proof system is a natural deduction calculus, and all rules are motivated to work well with backwards-reasoning. The type of proof trees is given by the following inductive family of types:

```
inductive prf :
    set (formula L) → formula L → Type u
| axm Γ A : A ∈ Γ → prf Γ A
| impI Γ A B : prf (insert A Γ) B →
    prf Γ (A ⟹ B)
| impE Γ A B : prf Γ (A ⟹ B) → prf Γ A →
    prf Γ B
| falsumE Γ A : prf (insert ~A Γ) ⊥ → prf Γ A
| allI Γ A : prf ((λ f, f ↑ 1) '' Γ) A →
    prf Γ (∀' A)
| allE₂ Γ A t : prf Γ (∀' A) →
    prf Γ (A[t // 0])
| ref Γ t : prf Γ (t ≃ t)
| subst₂ Γ s t f : prf Γ (s ≃ t) →
    prf Γ (f[s // 0]) → prf Γ (f[t // 0])
```

In `allI` the notation `(λ f, f ↑ 1) '' Γ` means lifting all free variables in Γ by one. A term of type `prf Γ A`, denoted `Γ ⊢ A`, is a proof tree encoding a derivation of A from Γ. We also define provability as the proposition stating that a proof tree exists.

```
def provable (Γ : set (formula L))
  (f : formula L) : Prop := nonempty (prf Γ f)
```

Our current formalization does not use the data of proof trees in an essential way, but we defined them so that we

can define manipulations on proof trees (like detour elimination) in future projects. Besides Boolean-valued semantics (Section 3), we also formalize ordinary first-order semantics, and our work includes a formalization of the completeness (and compactness) theorems using Henkin term models.

2.2 ZFC

Usually, the language of set theory has one binary relation symbol and no function symbols. To make the language easier to work with, and to concisely formulate the continuum hypothesis, we conservatively extend ZFC with the following function symbols: the empty set $\varnothing$, ordered pairing $(-,-)$, the natural numbers ω, power set $\mathcal{P}(-)$ and union $\bigcup(-)$. This gives a conservative extension of the regular theory of ZFC, because these function symbols are all definable.

In Figure 1 we have listed all the axioms of ZFC written using names variables (the formalization uses de Bruijn variables). We also include the definition of ordinal, which is used in the axiom of infinity. Note that `epsilon_wellfounded` follows for every set from the axiom of regularity, but we add it for the sake of completeness. The only axiom scheme is `axiom_of_collection` which ranges over all formulas φ `(x,y,p)` with (at most) `n+2` free variables, where `p` is a vector of length `n`.

Now CH is defined to be the sentence

$$\mathrm{CH} := \forall x, \mathrm{Ord}(x) \Rightarrow x \leq \omega \vee \mathcal{P}(\omega) \leq x,$$

where $x \leq y$ means that there is a surjection from a subset of y to x. In code, we have:

```
def CH_formula : formula L_ZFC :=
∀' (is_ordinal ⟹
  leq_f[omega_t//1] ⊔ leq_f[Powerset_t omega_t//0])
```

The substitutions ensure that the formulas are applied to the correct arguments, and $\sqcup$ is notation for disjunction.

3 Boolean-Valued Semantics

A **complete Boolean algebra** is a Boolean algebra $\mathbb{B}$ with additional operations infimum ($\bigsqcap$) and supremum ($\bigsqcup$) of any subset of $\mathbb{B}$. We use $\sqcap, \sqcup, \Longrightarrow, \top$, and $\bot$ to denote meet, join, material implication, top, and bottom. For more details on complete Boolean algebras, we refer the reader to the textbook of Halmos-Givant [13].

Definition 3.1. Fix a language L and a complete Boolean algebra $\mathbb{B}$. A **$\mathbb{B}$-valued structure** (or `bStructure L 𝔹`) is a type M equipped with the following.

- for every n-ary function symbol in a map $M^n \to M$;
- for every n-ary relation symbol a map $M^n \to \mathbb{B}$;
- a function $\approx : M \to M \to \mathbb{B}$ that is a Boolean valued congruence relation. This means that e.g. $x \approx y \sqcap y \approx z \leq x \approx z$ and that

$$\bigsqcap_i x_i \approx y_i \leq f(\vec{x}) \approx f(\vec{y}).$$

There are similar conditions for reflexivity, symmetry and congruence for relation symbols.

Given a preterm `t` in the language, we can realize it in any $\mathbb{B}$-valued structure M. For this, we need to know the free variables in `t`. To do this conveniently with de Bruijn variables, we say that a (pre)term `t` is *bounded by* `l` if all free variables are less than `l` (i.e. all variables under `k` quantifiers are less than `k+l`). Given `t : preterm n` which is bounded by `l`, and a realization `v : vector M l` of the free variables, we define the realization $[\![t]\!]^v_M : M^n \to M$ by structural recursion on t.

For a formula φ we do the same: we define bounded (pre)formulas, and define an realization $[\![\varphi]\!]^v_M : M^n \to \mathbb{B}$ by structural recursion. If φ is a sentence, the realization in a structure is just an element of the Boolean algebra: $[\![\varphi]\!]_M : \mathbb{B}$.

Since the truth values in a Boolean-valued model live inside the Boolean algebra $\mathbb{B}$ instead of just being true or false, we have to take a little care when stating the soundness theorem for Boolean-valued models. Usually, a soundness theorem states something like "if φ is provable from hypotheses in C then in every model where C holds, φ also holds." With Boolean truth-values, this is instead stated as an inequality of truth values.

Definition 3.2. For $\Gamma : \mathbb{B}$ and a $\mathbb{B}$-valued structure M we say that Γ **forces a sentence** φ **in** M, written $\Gamma \Vdash_M \varphi$, if $\Gamma \leq [\![\varphi]\!]_M$. We say that a set of sentences C **models** φ, written $C \models_{\mathbb{B}} \varphi$, if for all non-empty $\mathbb{B}$-valued structures M we have $\left(\bigsqcap_{\psi \in C} [\![\psi]\!]_M\right) \Vdash_M \varphi$.

Using this definition, we can now state the Boolean-valued soundness theorem:

```
theorem boolean_soundness {Γ : set (sentence L)}
    {φ : sentence L} : Γ ⊢ φ → Γ ⊨[𝔹] φ
```

The proof is a straightforward structural induction.

4 Boolean-Valued Models of Set Theory

4.1 The Aczel Encoding

Our starting point is the Aczel encoding of ZFC ([1–3]) into dependent type theory. This was implemented in Coq by Werner [46], and in Lean's mathlib by Carneiro [7]. The idea is to take a type universe `Type u` and imitate the cumulative hierarchy construction with an inductive type:

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

For an element `x = ⟨α, A⟩ : pSet`, the function `A` points to the elements of `x`. We can define the empty set as `∅ := ⟨ empty, empty.elim⟩ : pSet`. Note that `pSet` does not satisfy the axiom of extensionality. In order to obtain a model where the axiom of extensionality holds, we must quotient `pSet` by *extensional equivalence*:

```
def equiv : pSet → pSet → Prop
```

```
axiom_of_emptyset := ∀ x, x ∉ ∅
axiom_of_ordered_pairs := ∀ x y z w, (x, y) = (z, w) ↔ x = z ∧ y = w
axiom_of_extensionality := ∀ x y, (∀ z, (z ∈ x ↔ z ∈ y)) → x = y
axiom_of_union := ∀ u x, x ∈ ⋃ u ↔ ∃ y ∈ u, x ∈ y
axiom_of_powerset := ∀ z y, y ∈ P(z) ↔ ∀ x ∈ y, x ∈ z
axiom_of_infinity := ∅ ∈ ω ∧ (∀ x ∈ ω, ∃ y ∈ ω, x ∈ y) ∧ (∃ α, Ord(α) ∧ ω = α) ∧
  ∀ α, Ord(α) → (∅ ∈ α ∧ ∀ x ∈ α, ∃ y ∈ α, x ∈ y) → ω ⊆ α
axiom_of_regularity := ∀ x, x ≠ ∅ → ∃ y ∈ x, ∀ z ∈ x, z ∉ y
zorns_lemma := ∀ z, z ≠ ∅ → (∀ y, (y ⊆ z ∧ ∀ x₁ x₂ ∈ y, x₁ ⊆ x₂ ∨ x₂ ⊆ x₁) → (⋃y) ∈ z) →
  ∃ m ∈ x, ∀ x ∈ z, m ⊆ x → m = x
axiom_of_collection(φ) := ∀ p ∀ A, (∀ x ∈ A, ∃ y, φ(x,y,p)) →
  (∃ B, (∀ x ∈ A, ∃ y ∈ B, φ(x,y,p)) ∧ ∀ y ∈ B, ∃ x ∈ A, φ(x,y,p))

epsilon_transitive(z) := ∀ x, x ∈ z ⟹ x ⊆ z
epsilon_trichotomy(z) := ∀ x y ∈ z, x = y ∨ x ∈ y ∨ y ∈ x
epsilon_wellfounded(z) := ∀ x, x ⊆ z ⟹ x ≠ ∅ → ∃ y ∈ x, ∀ w ∈ x, w ∉ y
Ord(z) := epsilon_trichotomy(z) ∧ epsilon_wellfounded(z) ∧ epsilon_transitive(z)
```

Figure 1. Our formulation of ZFC.

```
| ⟨α,A⟩ ⟨β,B⟩ := (∀a, ∃b, equiv (A a) (B b)) ∧
  (∀b, ∃a, equiv (A a) (B b))
```

One can then define membership from equivalence and check that modulo extensional equivalence, `pSet` is a model of ZFC.

4.2 Boolean-Valued Sets

We now want to generalize `pSet` to a Boolean-valued model of ZFC. We must give a $\mathbb{B}$-valued predicate interpreting the membership symbol $\in$. We will encode this information by extending each `⟨α,A⟩ : pSet` with an additional function `B : α → 𝔹`, which has the effect of attaching a *Boolean truth-value* to every element of `⟨α,A⟩`:

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

The $\mathbb{B}$-valued predicate `B` expresses that `A a ∈ ⟨α, A, B⟩` has truth value (at least) `B i`. For convenience, if `x : bSet 𝔹` and `x := ⟨α, A, B⟩`, we put `x.type := α`, `x.func := A`, `x.bval := B`.

One can also be led to this construction by considering the recursive *name*-construction from forcing, a key ingredient to building forcing extensions. Let $\mathbb{P}$ be a poset. From e.g. (Kunen [26], Definition IV.2.5):

Definition 4.1. A set τ is a $\mathbb{P}$-name iff τ is a relation and for all $\langle \sigma, p \rangle \in \tau$ we have that σ is a $\mathbb{P}$-name and $p \in \mathbb{P}$.

In particular, if $\mathbb{P}$ is the singleton poset, then a $\mathbb{P}$-name is merely a set of $\mathbb{P}$-names, in the same way that a term of type `pSet` is a type-indexed collection of terms of type `pSet`. Reversing this observation, we can replace $\mathbb{P}$ with a complete Boolean algebra $\mathbb{B}$ and generalize the definition of `pSet.mk` with a third field, so that as in the case of $\mathbb{P}$-names, every element of a set is assigned an element (a "Boolean truth-value") of $\mathbb{B}$, again giving us `bSet 𝔹`. Thus, `bSet 𝔹` should be thought of as the type of $\mathbb{B}$-names.

Boolean-Valued Equality and Membership We can define Boolean-valued equality and membership analogously to the definitions in `pSet`. To do this, we translate quantifiers and connectives into operations on $\mathbb{B}$:

```
def bv_eq : bSet 𝔹 → bSet 𝔹 → 𝔹
| ⟨α, A, B⟩ ⟨α′, A′, B′⟩ :=
  (⨅a, B a ⟹ ⨆a′, B′ a′ ⊓ bv_eq (A a) (A′ a′)) ⊓
  (⨅a′, B′ a′ ⟹ ⨆a, B a ⊓ bv_eq (A a) (A′ a′))
```

We abbreviate `bv_eq` with the infix operator $=^{\mathsf{B}}$. It is now easy to define $\mathbb{B}$-valued membership, which we denote by $\in^{\mathsf{B}}$.

```
def mem : bSet 𝔹 → bSet 𝔹 → 𝔹
| x ⟨α, A, B⟩ := ⨆a, B a ⊓ x =ᴮ A a
```

While standard treatments of Boolean-valued models of ZFC mutually define equivalence and membership so that the axiom of extensionality follows definitionally ([4], [17]), the induction principle given by the non-mutual definition is easier to work with in our formalization.

4.3 The Fundamental Theorem of Forcing

The fundamental theorem of forcing for Boolean-valued models [17] states that for any complete Boolean algebra $\mathbb{B}$, the type `bSet 𝔹` forms a Boolean-valued model of ZFC.

We mostly follow Bell [4] for the verification of the ZFC axioms in `bSet 𝔹`. Although most of the argument is routine,

we describe some aspects of `bSet 𝔹` which are revealed by this verification.

Notably, we can define subsets of a set `x : bSet 𝔹` by just modifying `x.bval`. This gives a nice definition of powerset:

Definition 4.2. Fix a $\mathbb{B}$-valued set `x = ⟨α, A, b⟩` and χ `: α → 𝔹` be a function. We define the $\mathbb{B}$-valued set $\widetilde{\chi}$ as $\langle \alpha, A, \chi \rangle$. The **powerset** $\mathcal{P}(x)$ of x is defined to be the $\mathbb{B}$-valued set

```
set_of_indicator χ := ⟨α → 𝔹, (λ χ, χ̃), (λ χ, χ̃ ⊆ᴮ x)⟩.
```

In particular, this gives an easy implementation of the axiom of comprehension (not just for interpretations of formulas, but for any $\mathbb{B}$-valued predicate on `bSet 𝔹` satisfying an appropriate $\mathbb{B}$-valued congruence lemma):

```
lemma bSet_axiom_of_comprehension (φ : bSet 𝔹 → 𝔹)
    (x : bSet 𝔹)
  (H_congr : B_ext φ) {Γ : 𝔹} :
  Γ ≤ ⨆ y, y ⊆ᴮ x ⊓ ⨅ z, z ∈ᴮ y ⇔ (z ∈ᴮ x ⊓ φ z)
```

Following Bell, we verify Zorn's lemma in `bSet 𝔹`. As is the case with `pSet`, establishing Zorn's lemma requires the use of a choice principle from the metatheory. This was the hardest 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. Using mixtures, one derives the *maximum principle*, which allows existentially quantified statements to be instantiated without changing their truth-value (so is essentially the axiom of choice):

```
lemma maximum_principle (φ : bSet 𝔹 → 𝔹)
(h_congr : B_ext φ) : ∃ u, (⨆(x:bSet 𝔹), φ x) = φ u
```

For example, if `x : bSet 𝔹` and φ is a $\mathbb{B}$-valued predicate, if we have that `⊤ ≤ ⨆ j : x.type, φ x`, there may not actually be some `j : x.type` which attains that supremum. However, the maximum principle ensures that a witness can be constructed via mixtures.

After we verify the (shallow) statements of all the axioms in `bSet 𝔹`, the last step is to construct a $\mathbb{B}$-valued `L_ZFC`-structure, called `V 𝔹`, on `bSet 𝔹`, and check that the interpretations of the axioms are $\top$. This amounts to proving that the deeply embedded statements correspond to the shallowly embedded statements. This is trivial for the axioms, since it is true by reflexivity, but takes more work for the axiom scheme of collection. This proves the following theorem.

```
theorem fundamental_theorem_of_forcing :
  ⊤ ⊩[V 𝔹] ZFC
```

4.4 Ordinals

Definition 4.3. We define the canonical map `check : pSet → bSet 𝔹` by

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

We write $\check{x}$ for `check x`, and call it a *check-name*. 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.[3]

In general, $\check{x}$ might have different properties than x, but Δ_0 properties (i.e. those definable with only bounded quantification) are always preserved. Importantly, `bSet 𝔹` thinks $\check{\omega}$ is ω. Notably, `ω : pSet` is defined separately from `ordinal.mk omega` (see below) as the finite von Neumann ordinals indexed by $\mathbb{N}$, so the underlying types of ω and $\check{\omega}$ are exactly $\mathbb{N}$.

The treatment of ordinals in `mathlib` associates a class of ordinals to every type universe, defined as isomorphism classes of well-ordered types. Lean's ordinals may be represented inside `pSet` by defining a map `ordinal.mk : ordinal → pSet` via transfinite recursion (indexing the von Neumann construction of ordinals). In pseudocode,

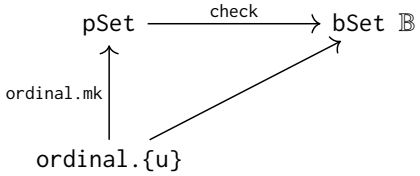
```
def ordinal.mk : ordinal → pSet
| 0 := ∅
| succ ξ := pSet.succ (ordinal.mk ξ)
            -- i.e. (mk ξ ∪ {mk ξ})
| is_limit ξ := ⋃ η < ξ, (ordinal.mk η)
```

Working internally to any model M of ZFC, we can define the class $\mathrm{Ord}(M)$ as the collection of transitive sets which are well-ordered by their membership relation. While `ordinal.mk` actually induces an order-isomorphism of `pSet`'s ordinals with Lean's ordinals, the map

```
check ∘ ordinal.mk : ordinal → bSet 𝔹
```

generally fails to surject onto `bSet 𝔹`'s ordinals (in general, these are mixtures of checked ordinals).

We summarize the relationship between the three "large" types currently in play:



We adopt the convention to spell out the name of Lean ordinals and cardinals, and use (checked) Hebrew letters for their (Boolean-valued) set-theoretic counterparts, e.g.

```
check (ordinal.mk (aleph 1)) = check (ℵ₁) = ℵ₁ˇ
```

We will freely conflate `pSet` ordinals with their underlying types, so e.g. `ν : ℵ₂` means `ν : ℵ₂.type`. (It is always true that the cardinality of `(ordinal.mk κ).type` is κ.) Since in general, $\widetilde{\aleph_1}$ is *not* what `bSet 𝔹` thinks is $\aleph_1$, we will use a superscript, e.g. $\aleph_k^{\mathbb{B}}$, to denote the internal alephs of `bSet 𝔹`.

[3] We were pleased to discover Lean's support for custom notation allowed us to declare the Unicode modifier character `U+030C` (ˇ) as a postfix operator for check.

5 Forcing

Our point of departure from conventional accounts of forcing with a poset $\mathbb{P}$ over a countable transitive model [25, 26], which use a generic filter to "evaluate" the $\mathbb{P}$-names to produce an ordinary model of ZFC, is to force with *Boolean-valued models* of ZFC instead. As first observed by Scott and Solovay [40], this obviates the need for countable transitive models, generic filters, or the truth and definability lemmas, and allows us to work only with the $\mathbb{B}$-names.

The cost of taking the $\mathbb{B}$-names at face value is that the calculus of the forcing relation [43], a key technical tool in usual forcing arguments, is replaced by the calculation of Boolean truth-values in $\mathbb{B}$. From the Boolean-valued perspective, forcing a sentence Φ in the language of ZFC means constructing some Boolean algebra $\mathbb{B}$ and a $\mathbb{B}$-valued model M of ZFC such that the truth value Φ^M of Φ is $\top$. We will always force over a type universe `Type u`, and our Boolean-valued models of ZFC are always of the form `bSet 𝔹` for some `𝔹 : Type u`. That $\mathbb{B}$ belongs to the "ground model" `Type u` is crucial for forcing, as specific choices of $\mathbb{B}$ will affect the structure of `bSet 𝔹` (and hence the truth-value of Φ).

In this section, we describe two forcing arguments, one for ¬CH and another for CH. Both follow roughly the same pattern. In both cases, we require the existence of a function; for ¬CH, an injection $\aleph_2 \hookrightarrow \mathcal{P}(\omega)$, and for CH, a surjection $\aleph_1 \twoheadrightarrow \mathcal{P}(\omega)$. We will construct a Boolean algebra $\mathbb{B}$ which encodes the construction (in `Type u`) of such a function F. Then $\mathbb{B}$ induces in `bSet 𝔹` an approximation $\widetilde{F}$ to such a function, which *a priori* is only between check-names. To finish the forcing argument, we must show that it suffices to work with $\widetilde{F}$. This requires a careful study of how truth-values are calculated in `bSet 𝔹`, and ultimately reduces to an analysis of how truth-values of ∀-∃ statements in `bSet 𝔹` can be *reflected back* to `Type u`, and a verification of a combinatorial condition on $\mathbb{B}$.

5.1 Regular Open Algebras

Definition 5.1. 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 $U = (U^\perp)^\perp$, or equivalently such that U is equal to the interior of the closure of U. $\mathrm{RO}(X)$ is equipped with the structure of a complete Boolean algebra, with $x \sqcap y := x \cap y$ and $x \sqcup y := ((x \cup y)^\perp)^\perp$ and $\neg x := x^\perp$ and $\bigsqcup x_i := ((\bigcup x_i)^\perp)^\perp$.

While forcing conditions usually present themselves as a poset instead of a complete Boolean algebra, any forcing poset can be represented as the dense suborder of a regular open algebra [30].

Definition 5.2. A **dense suborder** of $\mathbb{B}$ is a subset $\mathbb{P} \subseteq \mathbb{B}$ satisfying the following conditions: (1) for all $p \in \mathbb{P}$, $\bot < p$; (2) for all $\bot < b \in \mathbb{B}$, there exists a $p \in \mathbb{P}$ such that $p \leq b$.

We will use the following combinatorial conditions on $\mathbb{B}$ in our forcing arguments:

Definition 5.3. 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 $\mathcal{A} = \{a_i\}_i$ such that whenever $i \neq j, a_i \sqcap a_j = \bot$) has a countable image.

Definition 5.4. We say that $\mathbb{B}$ is **σ-closed** if there exists a dense suborder $\mathbb{P}$ of $\mathbb{B}$ such that every ω-indexed downwards chain $p_0 \geq \cdots \geq p_n \cdots$ in $\mathbb{P}$ has a lower bound p_ω in $\mathbb{P}$.

5.2 Cohen Forcing

As we have already seen in Definition 4.2, we construct the powerset of a $\mathbb{B}$-valued set `u : bSet 𝔹` using $\mathbb{B}$-valued indicator functions χ `: u.type →` $\mathbb{B}$. The basic strategy of Cohen forcing is to choose $\mathbb{B}$ such that for every $\nu : \aleph_2$, there is a canonical indicator function (a "Cohen real") $\chi_\nu : \mathbb{N} \to \mathbb{B}$. This is an external function (a member of a function type of `Type u`) which descends to an injective function $\widetilde{\aleph_2} \hookrightarrow \mathcal{P}(\omega)$ in `bSet 𝔹`.

To show that the injection $\widetilde{\aleph_2} \hookrightarrow \mathcal{P}(\omega)$ suffices to negate CH, we will show that if $\mathbb{B}$ has the CCC, then $\omega \prec \widetilde{\aleph_1} \prec \widetilde{\aleph_2}$, where $x \prec y$ means that there is no surjection from a subset of x to y. We then ensure that $\mathbb{B}$ has this property by applying a powerful combinatorial argument called the Δ-*system lemma.*

Definition 5.5. 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 ℙ_cohen : 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 that `f` is well-defined. While the members of the Cohen poset are usually defined as finite partial functions, we found that in practice `f` is 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). We chose this representation to make that information immediately accessible.

The Boolean algebra which we use for forcing ¬CH is

$$\mathbb{B}_{\mathsf{cohen}} := \mathrm{RO}(2^{\aleph_2 \times \mathbb{N}})$$

where we equip $2^{\aleph_2 \times \mathbb{N}}$ with the usual product space topology.

Definition 5.6. We define the **canonical embedding** of the Cohen poset into $\mathbb{B}_{\mathsf{cohen}}$ as follows:

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

That is, we send each $c : \mathbb{P}_{\text{cohen}}$ to all subsets satisfying the specification given by c. This is clopen, hence regular.

Crucially, the image of this embedding is a dense suborder of $\mathbb{B}_{\text{cohen}}$. This is essentially because the image of $\iota : \mathbb{P}_{\text{cohen}} \to \mathbb{B}_{\text{cohen}}$ *is* the standard basis for the product topology. Our chosen encoding of the Cohen poset also made it easier to perform this identification.

Definition 5.7. 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 $2^{\aleph_2 \times \mathbb{N}}$, denoted $\mathbf{P}_{(\nu,n)}$. Thus, we associate to ν the $\mathbb{B}$-valued indicator function $\chi_\nu : \mathbb{N} \to \mathbb{B}$ defined by $\chi_\nu(n) := \mathbf{P}_{(\nu,n)}$. By Definition 4.2, 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**.

Definition 5.7 gives us an $\aleph_2$-indexed family of Cohen reals. Converting this data into an injective function from $\widetilde{\aleph_2}$ to $\mathcal{P}(\mathbb{N})$ inside `bSet 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.

To finish negating CH, it suffices to show that $\omega \prec \widetilde{\aleph_1} \prec \widetilde{\aleph_2}$, i.e. that there is no surjection $\check{\omega} \twoheadrightarrow \widetilde{\aleph_1}$ and no surjection $\widetilde{\aleph_1} \twoheadrightarrow \widetilde{\aleph_2}$. We describe how we proved the latter claim; an identical argument can be used to show the former.

The strategy of the proof is to assume that there is a surjection $\widetilde{\aleph_1} \twoheadrightarrow \widetilde{\aleph_2}$. This surjectivity assumption is a Boolean-valued ∀-∃ statement about check-names, and we will *reflect* it into the metatheory, producing a ∀-∃ statement about the non-checked counterparts in `pSet`. We will then use the CCC, a combinatorial condition on $\mathbb{B}_{\text{cohen}}$, to show that the reflected ∀-∃ statement implies a contradiction.

Specifically, we use the following lemma, which is true for general $\mathbb{B}$:

```
lemma AE_of_check_larger_than_check {x y : pSet}
  (f : bSet 𝔹) {Γ : 𝔹} (H_nonzero : ⊥ < Γ)
  (H : Γ ≤ is_surj_onto x̌ y̌ f) (Hy : ∃ z, z ∈ y) :
  ∀ i : y.type, ∃ j : x.type,
  ⊥ < is_func f ⊓ pair (x.func j)̌  (y.func i)̌  ∈ᴮ f
```

Suppose that there is a surjection $\widetilde{\aleph_1} \twoheadrightarrow \widetilde{\aleph_2}$. Applying this lemma to $x := \widetilde{\aleph_1}$, $y := \widetilde{\aleph_2}$, we obtain a ∀-∃ statement in the metatheory to which we can apply Lean's axiom of choice to produce a function $g : \aleph_2 \to \aleph_1$. Since externally, we know that $\aleph_1 \prec \aleph_2$, it follows from the infinite pigeonhole principle that g must have an uncountable fiber over some $\nu < \aleph_1$. For every $\eta \in g^{-1}(\{\nu\})$, let A_η be the element of $\mathbb{B}_{\text{cohen}}$ given by the lemma, i.e.

```
(is_func f) ⊓ (pair (ℵ₁.func ν)̌ (ℵ₂.func η)̌ ∈ᴮ f).
```

Because each A_η has as a conjunct the knowledge that f is a function, for $\eta_1 \neq \eta_2$, A_{η_1} and A_{η_2} are incompatible, i.e. $A_{\eta_1} \sqcap A_{\eta_2} = \bot$. Since the lemma guarantees that each A_η is nonzero, the A_η form an uncountable antichain. Therefore, if $\mathbb{B}$ has the CCC, there is a contradiction. By Lemma 5.3, ¬CH is forced true in `bSet` $\mathbb{B}_{\text{cohen}}$.

In our formalization, we actually prove a more general version of this argument, replacing $\aleph_1$ and $\aleph_2$ with any two infinite regular cardinals $\kappa_1 < \kappa_2$.

CCC and the Δ-system lemma To show that $\mathbb{B}_{\text{cohen}}$ has the CCC, we formalize and then apply a general result in transfinite combinatorics called the Δ-*system lemma*. Though only briefly mentioned in [18], this was one of the most involved parts of our formalization of Cohen forcing, as it was a technical result in infinitary combinatorics. The details of the full argument are too technical to give here, so we omit the proofs in this section.

A family $(A_i)_i$ of sets is called a **Δ-system** if there is a set r, called the **root** such that whenever $i \neq j$ we have $A_i \cap A_j = r$. We write $c^{<\kappa}$ for the supremum of c^ρ for $\rho < \kappa$.

Lemma 5.1 (Δ-system lemma (Theorem 1.6, [26]))**.** *Let κ be an infinite cardinal and let $\theta > \kappa$ be regular, such that for all $\alpha < \theta$ we have $\alpha^{<\kappa} < \theta$. For any family $\{A_i\}_{i \in I}$ such that $|I| \geq \theta$ and for all i, $|A_i| < \kappa$, there is a subfamily of size θ which forms a Δ-system.*

The formalization closely follows the proof given in Kunen [26, Chapter 2, Theorem 1.6]. The proof involves tricky reasoning steps involving ordinals, which are common in infinitary combinatorics. It starts by assuming that without loss of generality $\bigcup_i A_i \subseteq \theta$, so that all the A_i are well-ordered, and by assuming that all A_i have the same order-type. These simplifying assumptions are harder to formalize, because that involves actually proving the general case from the special case. It also involves defining a sequence by transfinite recursion, while simultaneously proving that the sequence has certain properties (lies below θ).

In the formalization, the fact that the type of ordinals is a large type, i.e. lives one universe level higher than the types it is built from, causes difficulties. (These difficulties were also present earlier, because whenever we use e.g. "`ℵ₂.type`", we are actually referring to a nonconstructively chosen witness for the order type of all the ordinals less than `aleph 2`.) The reason is that the original proof heavily uses sets of ordinals, and taking their order types, but in Lean this would involve calculating in both `ordinal.{u}` and `ordinal.{u+1}`. Instead, we frequently work with well-orders of a given order type, instead sets of ordinals, to do all computations in `ordinal.{u}`.

Lastly, one must take care to formulate the Δ-system so that $\{A_i\}_i$ is an indexed families, instead of a collections of sets. Theorem 5.1 below does not follow conveniently from

the Δ-system lemma if it is formulated with a collection of sets. [26] is somewhat ambiguous which version is used.

Setting $\kappa = \omega$ and $\theta = \aleph_1$ in Lemma 5.1 yields:

Lemma 5.2. *Any uncountable family of finite sets has an uncountable subfamily forming a Δ-system.*

We say that a topological space has the CCC if every family of pairwise disjoint open sets is countable. The proof of the following can be found in [18].

Theorem 5.1. *For any family $(X_i)_{i\in I}$ of topological spaces, $\prod_{i\in I} X_i$ has the CCC if for every finite $J \subseteq I$, the product $\prod_{i\in J} X_i$ has the CCC.*

From Theorem 5.1 and the observation that 2^J has the CCC if J is finite, the result follows.

Lemma 5.3. $\mathbb{B}_{\text{cohen}}$ *has the CCC.*

5.3 Collapse Forcing

Whereas Cohen forcing creates a new injection $\widetilde{\aleph_2} \hookrightarrow \mathcal{P}(\omega)$, we can use *collapse forcing* to create a new surjection $F : \aleph_1^{\mathbb{B}} \twoheadrightarrow \mathcal{P}(\omega)$. Similarly to Cohen forcing, the strategy is to pick $\mathbb{B}$ such that there is a canonical $\mathbb{B}$-valued indicator function on $\widetilde{\aleph_1} \times \widetilde{\mathcal{P}(\omega)}$ representing the graph of a surjection $\widetilde{F}$. To show that $\widetilde{F}$ suffices to force CH, we must verify that our choice of $\mathbb{B}$ is σ-closed.

The formalization of collapse forcing is actually much more involved than the formalization of Cohen forcing. In Cohen forcing, we have to do relatively little work inside of `bSet` $\mathbb{B}$ itself besides proving basic properties of functions. The difficulty is concentrated in proving and applying the CCC, which mostly happens in the metatheory. Moreover, constructing the new function (and the rest of the argument) required no density arguments at all. This is because in order to force ¬CH, we only had to ensure there was *some* infinite cardinality between ω and $\mathcal{P}(\omega)$ (we did not determine exactly which internal aleph number $\widetilde{\aleph_1}$ was in `bSet` $\mathbb{B}$).

However, to force CH, the quantifiers are flipped and now we must exclude *all* cardinalities between ω and $\mathcal{P}(\omega)$. From cleverly choosing $\mathbb{B}$, the best we can do is to construct a surjection $\pi : \widetilde{\aleph_1} \twoheadrightarrow \widetilde{\mathcal{P}(\omega)}$, and we are forced to prove that $\widetilde{\aleph_1} = \aleph_1^{\mathbb{B}}$ and $\widetilde{\mathcal{P}(\omega)} = \mathcal{P}(\omega)$. This means we must define and construct $\aleph_1^{\mathbb{B}}$, entailing, for example, the development of the theory of ordinals internal to `bSet` $\mathbb{B}$. For comparison, our library on set theory in `bSet` $\mathbb{B}$ totalled 2723 LOC when we forced ¬CH, and grew to 7020 LOC after forcing CH.

Definition 5.8. We define $\mathbb{P}_{\text{collapse}}$ to be the poset of countable partial functions $\aleph_1 \to \mathcal{P}(\omega)$. The principal open sets

$$D_p := \{g : \aleph_1 \to \mathcal{P}(\omega) \mid g \text{ extends } p\}, \quad p \in \mathbb{P}_{\text{collapse}}$$

form the basis of a topology τ (finer than the product topology) on the function set $\mathcal{P}(\omega)^{\aleph_1}$. We put

$$\mathbb{B}_{\text{collapse}} := \text{RO}\left(\mathcal{P}(\omega)^{\aleph_1}, \tau\right).$$

Lemma 5.4. $\mathbb{B}_{\text{collapse}}$ *is σ-closed.*

Proof. We show that the collection of principal open sets $\mathcal{D} := \{D_p\}_p$ forms a dense subset of $\mathbb{B}_{\text{collapse}}$ such that every ω-indexed downwards chain in $\mathcal{D}$ has a lower bound in $\mathcal{D}$. Since $\mathcal{D}$ generates the topology, it is clearly a dense suborder. For an arbitrary ω-indexed downwards chain

$$D_{p_0} \supseteq D_{p_1} \supseteq \cdots \supseteq D_{p_n} \supseteq \cdots,$$

it follows from the definition of the principal open sets that $p_0 \subseteq p_1 \subseteq \cdots \subseteq p_n \subseteq \cdots$. Then put $p_\omega := \bigcup_i p_i$. Since the union of countable partial functions is a countable partial function, D_{p_ω} is a lower bound of $\{D_{p_i}\}_i$. □

Remark 5.1. As an implementation detail, in the formalization we *define* $\mathbb{P}_{\text{collapse}}$ to be the countable partial functions (in `Type u`) between `(ordinal.mk (aleph one) : pSet).type` and `(powerset omega : pSet).type`, so that $\mathbb{B}_{\text{collapse}}$-valued indicator functions on

```
ordinal.mk (aleph one) : pSet).type ×
    (powerset omega : pSet).type
```

are definitionally equal to $\mathbb{B}_{\text{collapse}}$-valued indicator functions on the underlying types of `check (ordinal.mk (aleph one))` and `check (powerset omega)`.

To specify the surjection $\widetilde{\aleph_1} \twoheadrightarrow \widetilde{\mathcal{P}(\omega)}$, we specify a subset (the graph of the function) of the powerset $\mathcal{P}(\widetilde{\aleph_1} \times \widetilde{\mathcal{P}(\omega)})$. In `bSet` $\mathbb{B}_{\text{collapse}}$, we can do this by specifying the indicator function χ_π of the graph of a function $\pi : \widetilde{\aleph_1} \to \widetilde{\mathcal{P}(\omega)}$ as follows: to an $\eta < \aleph_1$ and a subset $S \subseteq \mathcal{P}(\omega)$ (in `pSet`), we attach the *principal open* (comprising functions extending the singleton countable partial function $\{(\eta, S)\}$):

$$\chi_\pi(\eta, S) := D_{\{(\eta,S)\}} = \{g : \aleph_1 \to \mathcal{P}(\omega) \mid g(\eta) = S\}.$$

More generally, we formalize conditions over generic `x`, `y : pSet` and $\mathbb{B}$ for when a function `af : x.type → y.type → 𝔹` induces a surjection $\check{x} \to \check{y}$ in `bSet` $\mathbb{B}$. By definition, such a function always induces a relation on the product (in `bSet` $\mathbb{B}$) of `x` and `y`. Surjectivity is equivalent to `⨅ j, (⨆ i, af i j) = ⊤`, totality is equivalent to `⨅ i, (⨆ j, af i j) = ⊤`, and well-definedness follows from conditions:

```
(∀ i, ∀ j₁ j₂, j₁ ≠ j₂ → af i j₁ ⊓ af i j₂ ≤ ⊥)
(∀ i₁ i₂, ⊥ < (func x i₁) =ᴮ (func x i₂) → i₁ = i₂)
```

Both surjectivity and totality of χ_π require *density arguments*, where the definition of indexed supremum $(\bigsqcup x_i)$ in the regular open algebra as the regularization $((\bigcup x_i)^\perp)^\perp$ of the set-theoretic union plays a key role: the union of the truth values is not the entire space, but is only a dense open whose regularization is the entire space. In particular, the density argument for surjectivity crucially uses that $\aleph_1$ is uncountable while ω is countable.

To finish demonstrating that CH is true in `bSet` $\mathbb{B}_{\text{collapse}}$, it remains to check that $\widetilde{\mathcal{P}(\omega)} = \mathcal{P}(\omega)$ and $\widetilde{\aleph_1} = \aleph_1^{\mathbb{B}}$. There

are two major obstacles. The first is that to even formally state the latter equality, we must construct $\aleph_1^{\mathbb{B}}$ in `bSet 𝔹`. While the operation `bv_powerset` (Definition 4.2) gives a construction of the internal powerset of any `x : bSet 𝔹` (using $\mathbb{B}$-valued indicator functions, for any $\mathbb{B}$), $\aleph_1^{\mathbb{B}}$ is only specified as the least ordinal greater than ω, and does not admit as direct of a construction. We describe our construction of $\aleph_1^{\mathbb{B}}$ (as the Hartogs number of ω) in Section 5.4.

Now we must ensure that no new countable ordinals are added to $\aleph_1$ and that no new subsets of ω are added to $\mathcal{P}(\omega)$ in the passage via `check` from `pSet` to `bSet 𝔹`. We show this in Section 5.5 by proving that we can reflect functions with domain ω from `bSet 𝔹` to `pSet`.

5.4 Construction of $\aleph_1$

Instead of using the specification of $\aleph_1^{\mathbb{B}}$ as the least ordinal larger than ω with Cantor's theorem and using the well-foundedness of the ordinals to construct $\aleph_1$, we opt for a direct construction of $\aleph_1$, based on the well-known construction of $\aleph_1$ as the **Hartogs number** of ω [21].

We lay out the basic strategy. Recall that a term of type `bSet 𝔹` comprises three pieces of information: an indexing type α, an indexing function `A : α → bSet 𝔹`, and a truth-value function `B : α → 𝔹`.

1. We *define* the underlying type α for $\aleph_1^{\mathbb{B}}$ to be $\mathcal{P}(\omega \times \omega)$`.type`.
2. We define the truth-value function `B : α → 𝔹` to assign to any $R \subseteq \omega \times \omega$ the (truth-value of) the sentence, "there exists an ordinal η and an injection $f : \eta \hookrightarrow \omega$ such that R is the image of the membership relation of η under f."
3. Using the maximum principle (which is essentially AC), we define the indexing function A for $\aleph_1^{\mathbb{B}}$ by choosing, for every $R : \alpha$, a witness η_R such that R is the image of η under an injection into ω. That A surjects onto countable ordinals reduces to the fact that order-isomorphic ordinals must be equal.

Implementation details In the formalization, this strategy is implemented in three stages. First, the axiom of comprehension (Section 4.3) is applied to $\mathcal{P}(\omega \times \omega)$ to produce (what `bSet 𝔹` thinks is) the collection of all relations R on ω such that $B(R)$ holds. This combines steps 1 and 2 and produces a set `a1'_aux`. Then we *modify* the indexing function `a1'_aux.func` (by using the maximum principle) to point from R to a chosen witness η_R for R, producing `a1'`. Finally, since the ordinals 0 and 1 both have empty membership relations, it is unprovable in Lean whether `a1'` contains one or the other, so we add both manually, producing $\aleph_1^{\mathbb{B}}$.

Our implementation differs from the usual construction of Hartogs numbers by *starting* with the sub-well-orders of ω, rather than taking the class of countable ordinals and later showing it is a set. In this way we avoid performing a smallness argument, at the cost of using the axiom of choice to select witnesses. We remark that our construction does not use specific properties of ω and easily generalizes to construct the successor cardinal of any infinite set. Instead of using membership ($<$), we could have used subset ($\leq$) instead, which would avoid the intermediate `a1'`, but this would have made other parts of the proof more complex.

5.5 Function Reflection

Suppose given `y : pSet` and `f : bSet 𝔹` such that `bSet 𝔹` models that `f` is a function from ω to $\check{y}$. We say that `bSet 𝔹` **reflects `f`** if there exists a `g : pSet` such that `g` is a function from ω to `y` in `pSet`, and `bSet 𝔹` models that $\check{g} = f$. We say that `bSet 𝔹` **reflects countable functions** if it reflects all such `f`.

Lemma 5.5. *Let $\mathbb{B}$ be a complete Boolean algebra, and suppose that* `bSet 𝔹` *reflects countable functions. Then $\widetilde{\mathcal{P}(\omega)} = \mathcal{P}(\omega)$ and $\widetilde{\aleph_1} = \aleph_1^{\mathbb{B}}$.*

Proof. To see that $\widetilde{\aleph_1} \subseteq \aleph_1^{\mathbb{B}}$, let x be an arbitrary element of $\widetilde{\aleph_1}$. By definition x is equal to $\check{\eta}$ for some $\eta < \aleph_1$ in `pSet`. Since the ordinals and cardinals of `pSet` are isomorphic to Lean's ordinals and cardinals for `Type u`, η injects into ω (in `pSet`, and also at the level of indexing types). Since being an injective function is Δ_0, it is absolute for `check`, so $x = \check{\eta}$ injects into ω. Then, by definition of $\aleph_1^{\mathbb{B}}$ we have $x \in \aleph_1^{\mathbb{B}}$.[4]

To see that $\aleph_1^{\mathbb{B}} \subseteq \widetilde{\aleph_1}$, suppose towards a contradiction that this is not true; since the ordinals are well-ordered, this means that $\widetilde{\aleph_1} < \aleph_1^{\mathbb{B}}$, so by definition of $\aleph_1^{\mathbb{B}}$, there is a surjection $f : \omega \to \widetilde{\aleph_1}$. By assumption, this surjection can be lifted to a function $g : \omega \to \aleph_1$ in `pSet`, which can again be checked to be surjective, a contradiction.

Similarly, it is true for general $\mathbb{B}$ and any `x : pSet` that $\widetilde{\mathcal{P}(x)} \subseteq \mathcal{P}(\check{x})$, because indicator functions into `bool` naturally induce indicator functions to $\mathbb{B}$ (by composing with the canonical inclusion `bool → 𝔹`). Conversely, to show that $\mathcal{P}(\omega) \subseteq \widetilde{\mathcal{P}(\omega)}$, use the isomorphism $\mathcal{P}(\omega) \simeq \check{2}^{\check{\omega}}$ to reduce this to showing that $\check{2}^{\check{\omega}} \subseteq \widetilde{2^{\omega}}$, and then apply the assumption to an arbitrary element of $\check{2}^{\check{\omega}}$. □

It remains to show that $\mathbb{B}_{\text{collapse}}$ fulfills the assumptions of Lemma 5.5.

Lemma 5.6. `bSet` $\mathbb{B}_{\text{collapse}}$ *reflects countable functions.*

Proof. Fix y and f. It suffices to show that

```
f ∈ᴮ functions ω yˇ
  ≤ ⨆ (g : bSet 𝔹),
      g ∈ᴮ (functions omega y)ˇ  ⊓ g =ᴮ f
```

and by a density argument, it suffices to show that for every principal open D_p, for $D := D_p \cap f \in^{\mathbb{B}}$ functions $\omega\ \check{y}$,

```
⊥ < (⋃ g, D ⊓ g ∈ᴮ (functions omega y)ˇ  ⊓ g =ᴮ f)
```

[4] Note that this did not use our assumption, and holds for general $\mathbb{B}$. For a conventional proof in a set-theoretic metatheory, see e.g. [4]

It suffices to construct a single function $g : \omega \to y$ such that $\bot < D \sqcap \check{g} = f$. As with Cohen forcing, we will reflect a Boolean-valued $\forall$-$\exists$ statement into the metatheory, and then use a combinatorial property of $\mathbb{B}_{\text{collapse}}$ to strengthen it. The following lemma is true for general $\mathbb{B}$:

```
lemma AE_of_check_func_check (x y : pSet)
  {f : bSet 𝔹} {Γ : 𝔹}
  (H : Γ ≤ is_func' x̌ y̌ f) (H_nonzero : ⊥ < Γ) :
  ∀ (i : x.type), ∃ (j : y.type) (Γ' : 𝔹)
  (H_nonzero' : ⊥ < Γ') (H_le : Γ' ≤ Γ),
  Γ' ≤ (is_func' x̌ y̌ f) ⊓
  Γ' ≤ (pair (x.func i)̌ (y.func j)̌ ) ∈[𝔹] f
```

Recursively applying this lemma, we obtain $g_0, \ldots, g_n, \ldots$ such that

$$D \sqcap (0, g_0) \in^{\mathbb{B}} f > \cdots > D \sqcap \left(\prod_{k \le n} ((k, g_k) \in^{\mathbb{B}} f) \right) > \cdots > \bot.$$

The lower bound of this chain implies that the required lift of f is $g := \{(k, g_k)\}_{k \in \omega}$. For general $\mathbb{B}$, this lower bound might be $\bot$, but because $\mathbb{B}_{\text{collapse}}$ is σ-closed, we can shrink each term of the above chain into a dense suborder $\mathcal{D}$ such that all downward ω-indexed chains in $\mathcal{D}$ have nonzero intersection, so the intersection of the chain is indeed nonzero. □

Implementing this argument was one of the most technical parts of our formalization. At each step of the construction of the downwards chain, we must recursively apply a $\forall$-$\exists$ statement and use the axiom of choice to select two witnesses (with four side conditions), which are then used to simultaneously construct the downwards chain and the function `g : pSet`. This was implemented as a monolithic recursive function defined using Lean's equation compiler, with the required parts separated afterwards.

5.6 The Independence of CH

In Section 4.3 we showed that `bSet 𝔹` is a model of `ZFC`, which means that we can interpret the deeply-embedded statement of `CH_formula` into `bSet 𝔹`. We now verify that the deeply-embedded interpretations of `CH_formula` coincide with the shallow interpretations of CH.

As we have already observed, an easy consequence of Boolean-valued soundness is that a formula is unprovable if its negation has a model. Thus, we have:

```
lemma unprovable_of_model_neg {C : Theory L}
  {f : sentence L} (S : bStructure L 𝔹)
  (H_model : ⊤ ⊩[S] C) [H_nonempty : nonempty S]
  {Γ : 𝔹} (H_nonzero : (⊥ : 𝔹) < Γ)
  (H : Γ ⊩[S] ~f) : ¬ (C ⊢' f)
lemma V_𝔹_cohen_models_neg_CH :
  ⊤ ⊩[V 𝔹_cohen] ~CH_formula
lemma V_𝔹_collapse_models_CH :
  ⊤ ⊩[V 𝔹_collapse] CH_formula
```

Combining these results yields

```
theorem CH_unprv : ¬ (ZFC ⊢' CH_formula)
theorem neg_CH_unprv : ¬ (ZFC ⊢' ~CH_formula)
```

and the independence of CH follows.

```
def independent (T : Theory L) (f : sentence L) :=
¬ T ⊢' f ∧ ¬ T ⊢' ~f
theorem independence_of_CH : independent ZFC CH_f :=
by finish [independent, CH_unprv, neg_CH_unprv]
```

6 Automation and Metaprogramming

A key feature of Lean is that it is its own metalanguage [12], allowing for seamless in-line definitions of custom tactics (and modifications of existing ones). This was an invaluable asset, we were able to rapidly develop a custom tactic library for simulating natural-deduction style proofs in complete Boolean algebras (Section 6.1) and automating equality reasoning in those proofs (Section 6.2).

6.1 Simulating Natural Deduction Proofs in Complete Boolean Algebras

As stressed by Scott [39], "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 (see Section 3) is true: one can just replay natural deduction proofs in arbitrary complete Boolean algebras, not just `Prop`. We use Lean's metaprogramming to expose natural deduction-style tactics to the user for the purpose of proving inequalities in complete Boolean algebras. (One thinks of the $\le$ symbol in an inequality of Boolean truth-values as a turnstile in a proof state). An immediate challenge which arises is being able to reason about assumptions (to the left of the turnstile) modulo associativity and commutativity. For example, the natural-deduction version of this statement should simply be `by assumption`:

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

but with a naive approach, one must manually unwrap and permute the arguments of the nested $\sqcap$s. Our solution is to piggyback on the tactic monad's AC-invariant handling of hypotheses in the tactic state, by applying the *Yoneda lemma* for posets:

```
lemma poset_yoneda {β} [partial_order β] {a b : β}
  (H : ∀ Γ : β, Γ ≤ a → Γ ≤ b) : a ≤ b
```

With a little custom automation, our first example nearly becomes "`by assumption`"

```
example {a b c d e f g : 𝔹} :
  (d ⊓ e) ⊓ (f ⊓ g ⊓ ((b ⊓ a) ⊓ c)) ≤ a :=
```

```
by { tidy_context, assumption }
/-  Goal state before `assumption`:
  [...]
  H_right_right_left_left : Γ ≤ b,
  H_right_right_left_right : Γ ≤ a
  ⊢ Γ ≤ a  -/
```

In this example, `tidy_context` combines an application of `poset_yoneda` with a call to the simplifier to split hypotheses of the form $\Gamma \le a_1 \sqcap a_2 \sqcap \ldots a_n$ into $\Gamma \le a_1$, $\Gamma \le a_2$, $\ldots$, $\Gamma \le a_n$.

With more sophisticated tricks, such as coercing assumptions of the form $(\Gamma \le a \Longrightarrow b)$ to functions $\Gamma \le a \to \Gamma \le b$, automated propagation of change-of-variables ("context-specialization", see [18] for more details), and automatically casing on disjunctions $\Gamma \le a \sqcup b$, it is even possible to write a Boolean-valued tableaux prover `bv_tauto`:

```
example {a b c : 𝔹} :
  (a ⟹ b) ⊓ (b ⟹ c) ≤ a ⟹ c :=
  by { tidy_context, bv_tauto }
```

Compare this with a more conventional proof, where we even have the deduction theorem and modus ponens available as lemmas:

```
example {β : Type*} [complete_boolean_algebra β]
  {a b c : β} :
 ( a ⟹ b ) ⊓ ( b ⟹ c ) ≤ a ⟹ c :=
begin
  rw [ ← deduction, inf_comm, ← inf_assoc ],
  transitivity b ⊓ (b ⟹ c),
    { refine le_inf _ _,
      { apply inf_le_left_of_le, rw inf_comm,
        apply mp },
      { apply inf_le_right_of_le, refl }},
    { rw inf_comm, apply mp }
end
```

It would have been possible to go further and even write a custom tactic state, as was done for temporal logic in Unit-B [22] or for Lean's SMT-mode framework, such that the machinery for handling the ambient context Γ is completely hidden. However, we judged the benefits of this to be mostly cosmetic, and we leave more sophisticated implementations for future work.

6.2 Boolean-valued Equality Reasoning

Congruence Closure on Quotient Types Another benefit of applying `poset_yoneda` and using context variables Γ throughout the formalization is that this approach exposes a canonical poset of setoids on `bSet 𝔹` induced by $\mathbb{B}$-valued equality: for every $\Gamma : \mathbb{B}$ the relation $\lambda\ x\ y, \Gamma \le x =^{\mathbb{B}} y$ is an equivalence relation on `bSet 𝔹`.

Since Lean natively supports quotient types, then as soon as the only task remaining is to perform equality reasoning, we can quotient by the appropriate setoid and simply call `cc`; this is easy to automate with a custom tactic `bv_cc`. We can add support for any predicate satisfying an appropriate $\mathbb{B}$-valued congruence lemma, although we currently add support for individual predicates by hand:

```
example {x₁ y₁ x₂ y₂ : bSet 𝔹} {Γ}
  (H₁ : Γ ≤ x₁ ∈ᴮ y₁) (H₂ : Γ ≤ x₁ =ᴮ x₂)
  (H₂ : Γ ≤ y₁ =ᴮ y₂) : Γ ≤ x₂ ∈ᴮ y₂ := by bv_cc
```

Discharging Congruence Lemmas Rewriting along a $\mathbb{B}$-valued equality is the same as rewriting in the appropriate setoid parametrized by the current context Γ, so that the motive must satisfy an appropriate *congruence lemma* `h_congr` with respect to the equivalence relation:

```
lemma bv_rw {x y : bSet 𝔹} {Γ : 𝔹}
  (H : Γ ≤ x =ᴮ y) {φ : bSet 𝔹 → 𝔹}
  {h_congr : ∀ x y, x =ᴮ y ⊓ φ x ≤ φ y}
  {H_new : Γ ≤ φ y} : Γ ≤ φ x
```

We alias the type of `h_congr`, and add a database of `@[simp]` lemmas expressing that congruence lemmas are preserved by first-order logical operations:

```
def B_ext (φ : bSet 𝔹 → 𝔹) : Prop :=
∀ x y, x =ᴮ y ⊓ φ x ≤ φ y
@[simp] lemma B_ext_infi {ι} {φ : ι → (bSet 𝔹 → 𝔹)}
  {h : ∀ i, B_ext (φ i)} : B_ext (λ x, ⨅i, φ i x)
```

Furthermore, `simp` is able to handle recursive applications of these lemmas on its own, allowing most congruence lemma proof obligations to be automatically discharged:

```
example {w : bSet 𝔹} :
  (let φ := λ x, ⨅ z, z ∈ᴮ w ⊓ z ⊆ᴮ x ⊓ x ⊆ᴮ z
  in B_ext φ) := by simp
```

7 Conclusions and Future Work

Interestingly, we never used transfinite recursion for developing elementary set theory in `pSet` and `bSet 𝔹`. Indeed, the prevalence of transfinite recursion in traditional presentations of set theory is only a consequence of the use of transfinite recursion in the traditional definitions of V and $V^{\mathbb{B}}$. By instead encoding V and $V^{\mathbb{B}}$ as inductive types which expose $\in$-induction as their native induction principle, we completely eliminate transfinite induction from this part of our formalization.

Our consistency proof of CH is very different from the traditional proof, due to Gödel, which shows that the constructible universe L satisfies GCH. An obvious path to constructing L is to define the definable powerset operation with an inductive predicate on `pSet` whose constructors encode the nine Gödel operations, and to then build the constructible hierarchy by transfinite recursion. It is interesting to consider whether there is a definition of L in the same spirit as `pSet` which completely avoids transfinite induction.

We also want to formalize the conservativity of ZFC over the usual presentation in the language $\{\in\}$, by proving more generally that extending a language with definable function symbols is conservative. Furthermore, while formulas with de Bruijn indices enjoy pleasant theoretical properties, they are difficult to write and debug by hand. It should be possible with Lean's metaprogramming to write a custom parser from formulas with named variables.

Although our custom automation saved a considerable amount of work, much of it is only an approximation to a more principled approach by *reflection*. The natural deduction and equality reasoning tactics in Section 6.1 and Section 6.2 make it easier to manually replay a first-order proof of a theorem of ZFC in `bSet` $\mathbb{B}$, but the Boolean-valued soundness theorem automatically performs this replay for a deeply-embedded first-order proof tree. Ideally, automation would reify a $\mathbb{B}$-valued goal to the corresponding first-order statement, discharge it by an ATP, encode the solution in our deeply-embedded proof system, then apply soundness. Alternately, one could perform proof transfer via the completeness theorem, proving a first-order goal in an arbitrary ordinary model of ZFC first, then applying $\mathbb{B}$-valued soundness to the proof tree gotten by completeness. The advantage to this approach is that a proof would only be computed once, then reused in any model, ordinary or $\mathbb{B}$-valued, whereas in our formalization, we occasionally had to prove the same statement separately in `pSet` and `bSet` $\mathbb{B}$.

Besides the construction of L, the consistency of GCH can also be shown by an iterated forcing argument. Our current implementation of forcing should extend without too much difficulty to iterated forcing with Boolean-valued models. There are also many generalizations of the consistency of ¬CH. An interesting challenge could be Easton's theorem, which states that on regular cardinals the function $\kappa \mapsto 2^\kappa$ can be any monotone function not contradicting König's Theorem ($\kappa < \mathrm{cf}(2^\kappa)$) [11].

Our work only marks the beginning of an integration of formal methods with modern set theory. Since Cohen, increasingly sophisticated forcing arguments have been used to produce a vast hierarchy of independence and relative consistency results. The challenge to proof engineers is to develop libraries and automation that can uniformly handle them, so that the manipulation of forcing notions and forcing extensions in a proof assistant becomes as routine as manipulating objects in an algebraic hierarchy is today. One place to start would be to develop a good interface for forcing with posets, and for transferring arguments along the equivalence to Boolean-valued models. One could develop a typeclass hierarchy of combinatorial conditions on forcing notions, and similarly for the relative consistency strengths of extensions to ZFC. As the next challenge to formalizers, we propose the classical result of Shelah [42] on the independence of Whitehead's problem, the proof of which combines the consistency of the ZFC + (V = L) with the consistency of Martin's axiom [28] over ZFC + ¬CH to resolve a conjecture in abstract algebra.

Acknowledgments

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

The authors gratefully acknowledge the support by the Alfred P. Sloan Foundation, Grant No. G-2018-10067.

References

[1] Peter Aczel. 1978. The type theoretic interpretation of constructive set theory. In *Logic Colloquium*, Vol. 77. 55–66.

[2] Peter Aczel. 1982. The type theoretic interpretation of constructive set theory: choice principles. In *Studies in Logic and the Foundations of Mathematics*. Vol. 110. Elsevier, 1–40.

[3] Peter Aczel. 1986. The type theoretic interpretation of constructive set theory: inductive definitions. In *Studies in Logic and the Foundations of Mathematics*. Vol. 114. Elsevier, 17–49.

[4] John L Bell. 2011. *Set theory: Boolean-valued models and independence proofs*. Vol. 47. Oxford University Press.

[5] Stefan Berghofer. 2007. First-Order Logic According to Fitting. *Archive of Formal Proofs* (Aug. 2007). http://isa-afp.org/entries/FOL-Fitting.html, Formal proof development.

[6] Georg Cantor. 1878. Ein Beitrag zur Mannigfaltigkeitslehre. *Journal für die reine und angewandte Mathematik* 84 (1878), 242–258.

[7] Mario Carneiro. 2019. The type theory of Lean. (2019). In preparation (https://github.com/digama0/lean-type-theory/releases).

[8] Paul J Cohen. 1964. The independence of the continuum hypothesis. *Proceedings of the National Academy of Sciences* 50, 6 (1964), 1143–1148.

[9] Paul J Cohen. 1964. The independence of the continuum hypothesis, II. *Proceedings of the National Academy of Sciences* 51, 1 (1964), 105.

[10] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. 2015. The Lean Theorem Prover (System Description). In *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings (Lecture Notes in Computer Science)*, Amy P. Felty and Aart Middeldorp (Eds.), Vol. 9195. Springer, 378–388. https://doi.org/10.1007/978-3-319-21401-6_26

[11] William B Easton. 1970. Powers of regular cardinals. *Annals of mathematical logic* 1, 2 (1970), 139–178.

[12] Gabriel Ebner, Sebastian Ullrich, Jared Roesch, Jeremy Avigad, and Leonardo de Moura. 2017. A Metaprogramming Framework for Formal Verification. *Proc. ACM Program. Lang.* 1, ICFP, Article 34 (Aug. 2017), 29 pages. https://doi.org/10.1145/3110278

[13] Steven Givant and Paul Halmos. 2008. *Introduction to Boolean algebras*. Springer Science & Business Media.

[14] Kurt Gödel. 1938. The consistency of the axiom of choice and of the generalized continuum-hypothesis. *Proceedings of the National Academy of Sciences* 24, 12 (1938), 556–557.

[15] Emmanuel Gunther, Miguel Pagano, and Pedro Sánchez Terraf. 2018. First steps towards a formalization of Forcing. *CoRR* abs/1807.05174 (2018). arXiv:1807.05174 http://arxiv.org/abs/1807.05174

[16] Emmanuel Gunther, Miguel Pagano, and Pedro Sánchez Terraf. 2019. Mechanization of Separation in Generic Extensions. *CoRR* abs/1901.03313 (2019). arXiv:1901.03313 http://arxiv.org/abs/1901.03313

[17] Joel David Hamkins and Daniel Evan Seabold. 2012. Well-founded Boolean ultrapowers as large cardinal embeddings. *arXiv preprint*

arXiv:1206.6075 (2012).

[18] Jesse Michael Han and Floris van Doorn. 2019. A Formalization of Forcing and the Unprovability of the Continuum Hypothesis. In *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*. 19:1–19:19. https://doi.org/10.4230/LIPIcs.ITP.2019.19

[19] John Harrison. 1998. Formalizing Basic First Order Model Theory. In *Theorem Proving in Higher Order Logics, 11th International Conference, TPHOLs'98, Canberra, Australia, September 27 - October 1, 1998, Proceedings (Lecture Notes in Computer Science)*, Jim Grundy and Malcolm C. Newey (Eds.), Vol. 1479. Springer, 153–170. https://doi.org/10.1007/BFb0055135

[20] John Harrison. 2009. *Handbook of Practical Logic and Automated Reasoning*. Cambridge University Press.

[21] Friedrich Hartogs. 1915. Über das Problem der Wohlordnung. *Math. Ann.* 76, 4 (1915), 438–443.

[22] Simon Hudon, Thai Son Hoang, and Jonathan S. Ostroff. 2015. The Unit-B method: refinement guided by progress concerns. *Software & Systems Modeling* 15 (2015), 1091–1116.

[23] Joe Hurd and Thomas F. Melham (Eds.). 2005. *Theorem Proving in Higher Order Logics, 18th International Conference, TPHOLs 2005, Oxford, UK, August 22-25, 2005, Proceedings*. Lecture Notes in Computer Science, Vol. 3603. Springer. https://doi.org/10.1007/11541868

[24] Danko Ilik. 2010. *Constructive completeness proofs and delimited control*. Ph.D. Dissertation. Ecole Polytechnique X.

[25] Thomas Jech. 2013. *Set theory*. Springer Science & Business Media.

[26] Kenneth Kunen. 1980. *Set theory*. Studies in Logic and the Foundations of Mathematics, Vol. 102. North-Holland Publishing Co., Amsterdam-New York. xvi+313 pages.

[27] Yu I Manin. 2009. *A course in mathematical logic for mathematicians*. Vol. 53. Springer Science & Business Media.

[28] Donald A Martin and Robert M Solovay. 1970. Internal Cohen extensions. *Annals of Mathematical Logic* 2, 2 (1970), 143–178.

[29] The mathlib Community. 2019. The Lean mathematical library. *arXiv e-prints*, Article arXiv:1910.09336 (Oct 2019), arXiv:1910.09336 pages. arXiv:cs.LO/1910.09336

[30] Justin Tatch Moore. 2019. The method of forcing. *arXiv preprint arXiv:1902.03235* (2019).

[31] Russell O'Connor. 2005. Essential Incompleteness of Arithmetic Verified by Coq, See [23], 245–260. https://doi.org/10.1007/11541868_16

[32] Lawrence C. Paulson. 1993. Set Theory for Verification: I. From Foundations to Functions. *J. Autom. Reasoning* 11, 3 (1993), 353–389. https://doi.org/10.1007/BF00881873

[33] Lawrence C. Paulson. 2002. The Reflection Theorem: A Study in Metatheoretic Reasoning. In *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings (Lecture Notes in Computer Science)*, Andrei Voronkov (Ed.), Vol. 2392. Springer, 377–391. https://doi.org/10.1007/3-540-45620-1_31

[34] Lawrence C. Paulson. 2008. The Relative Consistency of the Axiom of Choice - Mechanized Using Isabelle/ZF. In *Logic and Theory of Algorithms, 4th Conference on Computability in Europe, CiE 2008, Athens, Greece, June 15-20, 2008, Proceedings (Lecture Notes in Computer Science)*, Arnold Beckmann, Costas Dimitracopoulos, and Benedikt Löwe (Eds.), Vol. 5028. Springer, 486–490. https://doi.org/10.1007/978-3-540-69407-6_52

[35] Lawrence C. Paulson and Krzysztof Grabczewski. 1996. Mechanizing Set Theory. *J. Autom. Reasoning* 17, 3 (1996), 291–323. https://doi.org/10.1007/BF00283132

[36] Tom Ridge and James Margetson. 2005. A Mechanically Verified, Sound and Complete Theorem Prover for First Order Logic, See [23], 294–309. https://doi.org/10.1007/11541868_19

[37] Anders Schlichtkrull. 2018. *Formalization of logic in the Isabelle proof assistant*. Ph.D. Dissertation. Technical University of Denmark.

[38] Dana Scott. 1967. A Proof of the Independence of the Continuum Hypothesis. *Theory of Computing Systems* 1, 2 (1967), 89–111.

[39] Dana Scott. 2008. The Algebraic Intepretation of Quantifiers: intuitionistic and classical. *Andrzej Mostowski and Foundational Studies* (2008), 289–312.

[40] Dana Scott and Robert Solovay. 1967. Boolean algebras and forcing. (1967). Unpublished manuscript.

[41] Natarajan Shankar. 1997. *Metamathematics, machines and Gödel's proof*. Vol. 38. Cambridge University Press.

[42] Saharon Shelah. 1974. Infinite abelian groups, Whitehead problem and some constructions. *Israel Journal of Mathematics* 18, 3 (1974), 243–256.

[43] Joseph R Shoenfield. 1971. Unramified forcing. In *Axiomatic set theory*, Vol. 13. AMS Providence, RI, 357–381.

[44] Sebastian Ullrich and Leonardo de Moura. 2019. Counting Immutable Beans: Reference Counting Optimized for Purely Functional Programming. arXiv:cs.PL/1908.05647

[45] Nik Weaver. 2014. *Forcing for mathematicians*. World Scientific.

[46] Benjamin Werner. 1997. Sets in types, types in sets. In *International Symposium on Theoretical Aspects of Computer Software*. Springer, 530–546.

[47] Freek Wiedijk. [n. d.]. Formalizing 100 theorems. http://www.cs.ru.nl/~freek/100/