

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

Jesse Michael Han¹

Department of Mathematics, University of Pittsburgh

<https://www.pitt.edu/~jmh288>

jessemichaelhan@gmail.com

Floris van Doorn

Department of Mathematics, University of Pittsburgh

<http://florisvandoorn.com/>

fpvdoorn@gmail.com

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 the failure of the continuum hypothesis in the resulting model.

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

Funding Both authors were supported by the Sloan Foundation, grant G-2018-10067.

Acknowledgements We 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 is 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.

In this paper we discuss the formalization of a Boolean-valued model of set theory where the continuum hypothesis fails. The work we describe is part of the Flypitch project, 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

¹ Corresponding author.

Calculus of Inductive Constructions and 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, 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 an entire universe of types; our encoding of first-order logic uses parametrized inductive types to equate type-correctness with well-formedness, eliminating the need for separate well-formedness proofs.

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.

We briefly outline the rest of the paper. 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 and the Boolean-valued soundness theorem. Section 3 discusses our construction of Boolean-valued models of set theory. Section 4 describes the formalization of the forcing argument and the construction of a suitable Boolean algebra for forcing \neg CH. Section 5 describes the formalization of some transfinite combinatorics. We conclude with a reflection on our formalization and an indication of future work.

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 $\text{ZFC} \vdash \text{CH}$, 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 \vee becomes join, \forall becomes infimum, etc.) into $\mathbf{2} = \{\text{true}, \text{false}\}$. The soundness theorem ensures that this homomorphism v sends a proof $\phi \vdash \psi$ to an inequality $v(\phi) \leq v(\psi)$. $\mathbf{2}$ may be replaced by any complete Boolean algebra \mathbb{B} , where the top and bottom elements \top, \perp 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 $\text{ZFC} \vdash \text{CH}$, 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 containing 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, properties which are definable with only bounded quantification are preserved; in particular, $V^{\mathbb{B}}$ thinks $\check{\aleph}$ is \aleph .

To force the negation of the continuum hypothesis, we use the Boolean algebra $\mathbb{B} := \text{RO}(2^{\aleph_2 \times \aleph})$ of regular opens of the Cantor space $2^{\aleph_2 \times \aleph}$. For each $\nu \in \aleph_2$, we associate the \mathbb{B} -valued characteristic function $\chi_\nu : \aleph \rightarrow \mathbb{B}$ by $n \mapsto \{f \mid f(\nu, n) = 1\}$. This induces what $V^{\mathbb{B}}$ thinks is a new subset $\check{\chi}_\nu \subseteq \aleph$, called a *Cohen real*, and furthermore, simultaneously performing this construction on all $\nu \in \aleph_2$ induces what $V^{\mathbb{B}}$ thinks is a function from $\check{\aleph}_2 \rightarrow \mathcal{P}(\aleph)$. 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 \aleph and $\mathcal{P}(\aleph)$. This is really the technical heart of the matter, and relies 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 $2^{\aleph_2 \times \aleph}$; we handle this with a general result in transfinite combinatorics called the Δ -system lemma.

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. While our objective is only to produce some model of ZFC satisfying certain properties, traditional presentations of forcing are careful to stay within the foundations of ZFC, emphasizing that all arguments may be performed internally inside a model of ZFC, etc., and it is not immediately clear what parts of the argument use that set-theoretic foundation in an essential way and require modification in the passage to type theory. As we will see, our formalization clarifies

some of these questions.

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, we follow Kunen ([?], Chapters 1 and 5).

Viewing the formalization The code blocks in this paper were taken directly from our formalization, but for the sake of formatting and readability, we sometimes omit or modify universe levels, type ascriptions, and casts. We refer the interested reader to our source code.² The forcing argument for the negation of CH is located in `forcing.lean`. In a Lean-aware editor such as Emacs or VSCode, the user is encouraged start at the theorem `neg_CH` and jump backwards to trace the dependencies of the proof.

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 :  $\mathbb{N} \rightarrow \text{Type } u$ ) (relations :  $\mathbb{N} \rightarrow \text{Type } u$ )
```

2.1 (Pre)terms, (pre)formulas

The main novelty of our implementation of first-order logic is the use of *partially applied* terms and formulas, encoded in a parametrized 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 :  $\mathbb{N} \rightarrow \text{Type } u$ 
| var {} :  $\forall (k : \mathbb{N}), \text{preterm } 0$ 
| func :  $\forall \{l : \mathbb{N}\} (f : L.functions \ l), \text{preterm } l$ 
| app :  $\forall \{l : \mathbb{N}\} (t : \text{preterm } (l + 1)) (s : \text{preterm } 0), \text{preterm } l$ 
```

We use de Bruijn indices to avoid variable shadowing. A member of `preterm n` is a partially applied term. If applied to n terms, it becomes a term. Every element of `preterm L 0` is a well-formed term. We use this encoding to avoid mutual or nested inductive types, since those are not too convenient to work with in Lean.

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

```
inductive preformula :  $\mathbb{N} \rightarrow \text{Type } u$ 
| falsum {} : preformula 0 -- notation  $\perp$ 
| equal (t1 t2 : term L) : preformula 0 -- notation  $\simeq$ 
| rel {l :  $\mathbb{N}$ } (R : L.relations l) : preformula l
| apprel {l :  $\mathbb{N}$ } (f : preformula (l + 1)) (t : term L) : preformula l
```

² <https://github.com/flypitch/flypitch>

```

177 | imp (f1 f2 : preformula 0) : preformula 0 -- notation ==>
178 | all (f : preformula 0) : preformula 0 -- notation '∀'
179 -- ¬ f := f ==> ⊥, notation '~f'
180 -- ∃ f := ~ ∀' ~f, notation '∃' f
181

```

182 A member of `preformula n` is a partially applied formula. If applied to `n` terms, it
 183 becomes a formula. Implication is the only binary connective. Since we use classical logic,
 184 we can define the other connectives from implication and falsum. Similarly, universal
 185 quantification is our only quantifier.

186 Our proof system is a natural deduction calculus, and all rules are motivated to work
 187 well with backwards-reasoning:

```

188
189
190 inductive prf : set (formula L) → formula L → Type u
191 | axm      {Γ A} (h : A ∈ Γ) : prf Γ A
192 | impI     {Γ} {A B} (h : prf (insert A Γ) B) : prf Γ (A ==> B)
193 | impE     {Γ} (A) {B} (h1 : prf Γ (A ==> B)) (h2 : prf Γ A) : prf Γ B
194 | falsumE  {Γ} {A} (h : prf (insert ~A Γ) ⊥) : prf Γ A
195 | allI     {Γ A} (h : prf Γ A) : prf Γ (∀' A)
196 | allE2   {Γ} A t (h : prf Γ (∀' A)) : prf Γ (A[t // 0])
197 | ref      (Γ t) : prf Γ (t ≈ t)
198 | subst2  {Γ} (s t f) (h1 : prf Γ (s ≈ t)) (h2 : prf Γ (f[s // 0])) :
199   prf Γ (f[t // 0])
200

```

201 A member of `prf Γ A` is a proof tree encoding a derivation of `A` from `Γ`. Note that `prf` is
 202 `Type`- instead of `Prop`-valued, so different members of `prf Γ A` are not definitionally equal.

203 2.2 Completeness

204 As part of our formalization of first-order logic, we completed a verification of the Gödel
 205 completeness theorem. Although our present development of forcing did not require it, we
 206 anticipate that it will be useful later to e.g. prove the downward Löwenheim-Skolem theorem for
 207 extracting countable transitive models. Like soundness, it also serves as a proof-of-concept
 208 and stress-test of our chosen encoding of first-order logic.

209 For our formalization, we chose the Henkin-style approach of constructing a canonical
 210 term model. In order to perform the argument, which normally involves modifying the
 211 language “in place” to iteratively add new constant symbols, we had to adapt it to type
 212 theory. Since our languages are represented by pairs of indexed types instead of sets, we
 213 cannot really modify them in-place with new constant symbols. Instead, at each step of
 214 the construction, we must construct an entirely new language in which the previous one
 215 embeds, and in the limit we must compute a directed colimit of types instead of a union.
 216 This construction induces similar constructions on terms and formulas, and completing the
 217 argument requires reasoning with all of them. As a result of our design decisions, only a
 218 few arguments required anything more than straightforward case-analysis and structural
 219 induction. The final statement makes no restrictions on the cardinality of the language.

220 2.3 Boolean-valued semantics for first-order logic

221 A **complete Boolean algebra** is a type \mathbb{B} equipped with the structure of a Boolean algebra
 222 and additionally operations `Inf` and `Sup` (which we write as \sqcap and \sqcup) 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.** Fix a language L and a complete Boolean algebra \mathbb{B} . A \mathbb{B} -valued structure is an instance of the following **structure**:

```

structure bStructure :=
  (carrier : Type u)
  (fun_map : ∀{n}, L.functions n → vector carrier n → carrier)
  (rel_map : ∀{n}, L.relations n → vector carrier n →  $\mathbb{B}$ )
  (eq : carrier → carrier →  $\mathbb{B}$ )
  (eq_refl : ∀ x, eq x x =  $\top$ )
  (eq_symm : ∀ x y, eq x y = eq y x)
  (eq_trans : ∀{x} y {z}, eq x y  $\sqcap$  eq y z  $\leq$  eq x z)
  (fun_congr : ∀{n} (f : L.functions n) (x y : vector carrier n),
     $\sqcap$ (map2 eq x y)  $\leq$  eq (fun_map f x) (fun_map f y))
  (rel_congr : ∀{n} (R : L.relations n) (x y : vector carrier n),
     $\sqcap$ (map2 eq x y)  $\sqcap$  rel_map R x  $\leq$  rel_map R y)

```

Above, “ \sqcap (map2 eq x y)” means “the infimum of the list whose i th entry is eq applied to $x[i]$ and $y[i]$ ”.

Note that Boolean-valued equality is not really an equivalence relation, but “ \mathbb{B} thinks it is”. One complication which then arises in Boolean-valued semantics is keeping track of the congruence lemmas 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.

2.4 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 of truth-values is **Prop**, 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:

```

lemma boolean_soundness { $\Gamma$  : set (formula L)} {A : formula L}
  (H :  $\Gamma \vdash A$ ) :  $\forall M, (\sqcap \gamma \in \Gamma, M[\gamma]) \leq M[A]$ 

```

Of course, we also formalized the ordinary soundness theorem. As a result of our design decisions, the proofs of both the ordinary and Boolean-valued soundness theorems were straightforward structural inductions.

3 Constructing Boolean-valued models of set theory

Throughout this section, we fix a universe level u and a complete Boolean algebra \mathbb{B} : **Type** u .

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
```

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 $x : \text{bSet } \mathbb{B}$ and $x := \langle \alpha, A, B \rangle$, we put $x.\text{type} := \alpha$, $x.\text{func} := A$, $x.\text{bval} := B$.

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 $=^B$. With equality in place, it is easy to define membership, by translating “ x is a member of y if and only if there exists a w indexed by the type of y such that $x = w$.” As with equality, we denote \mathbb{B} -valued membership with \in^B .

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


3.2 Automation and metaprogramming for reasoning in \mathbb{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 \leq 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 $a \ b : \mathbb{B}$, the identity $(a \Rightarrow b) \sqcap a \leq b$ could be proven by unfolding the definition of material implication, but it is really just modus ponens; similarly, given an indexed family $a : I \rightarrow \mathbb{B}$, $\bigsqcup i, a \ i \leq b \leftrightarrow \forall i, a \ i \leq b$ is just \exists -elimination.

Difficulties arise when the statements to be proved become only slightly more complicated. Consider the following example, which should be “by assumption”:

```


$$\forall a \ b \ c \ d \ e \ f \ g : \mathbb{B}, (d \sqcap e) \sqcap (f \sqcap g \sqcap ((b \sqcap a) \sqcap c)) \leq a$$


```

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

```


$$\forall a \ b \ c \ d : \mathbb{B}, (a \Rightarrow b) \sqcap c \sqcap (d \sqcap a) \leq b$$


```

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.³ 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** (Yoneda lemma for posets). *Let (P, \leq) be a partially ordered set. Let $a \ b : P$. Then $a \leq b$ if and only if $\forall \Gamma : P, \Gamma \leq a \rightarrow \Gamma \leq b$.*

This is a consequence 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 :  $\mathbb{B}$ } : (d  $\sqcap$  e)  $\sqcap$  (f  $\sqcap$  g  $\sqcap$  ((b  $\sqcap$  a)  $\sqcap$  c))  $\leq$  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:

```

example {a b c d e f g :  $\mathbb{B}$ } : (d  $\sqcap$  e)  $\sqcap$  (f  $\sqcap$  g  $\sqcap$  ((b  $\sqcap$  a)  $\sqcap$  c))  $\leq$  a :=
by {apply poset_yoneda, intros  $\Gamma$  H,
    simp only [le_inf_iff] at H /- turns H into conjunction -/,
    repeat {auto_cases} /- splits conjunction -/,
    assumption}
/- Goal state before `assumption`:

```

³ The deduction theorem in a Boolean algebra says that for all a, b and c , $a \sqcap b \leq c \iff a \leq b \Rightarrow c$.


```

362 [...]
363 H_right_right_left_left :  $\Gamma \leq b$ ,
364 H_right_right_left_right :  $\Gamma \leq a$ 
365  $\vdash \Gamma \leq a$  -/
366

```

367 A key feature of Lean is that it is its own metalanguage, allowing for seamless in-line
 368 definitions of custom tactics. This feature was an invaluable asset, as it allowed the rapid
 369 development of a custom tactic library for simulating natural-deduction style proofs inside
 370 \mathbb{B} after applying Lemma 3. The preprocessing steps before the call to `assumption` in the
 371 previous example are subsumed into a single tactic. Boolean-valued versions of natural
 372 deduction rules like \vee/\wedge -elimination, instantiation of existentials, implication introduction,
 373 and even basic automation were easy to write. The result is that the user is able to pretend,
 374 with absolute rigor, that they are simply writing proofs in first-order logic while calculations
 375 in the complete Boolean algebra are being performed under the hood.

376 One use-case where automation is crucial is context-specialization. For example, suppose
 377 that after preprocessing with `poset_yoneda`, the goal is $\Gamma \leq a \implies b$, and one would like
 378 to “introduce the implication”, adding $\Gamma \leq a$ to context and reducing the goal to $\Gamma \leq b$.
 379 This is impossible as stated. Rather, the deduction theorem lets us rewrite the goal to
 380 $\Gamma \sqcap a \leq b$, and now we may add $\Gamma \sqcap a \leq a$. So we may introduce the implication after
 381 all, but at the cost of specializing the context Γ to the smaller context $\Gamma' := \Gamma \sqcap a$. But
 382 now, in order for the user to continue the pretense that they are merely doing first-order
 383 logic, this change of variables must be propagated to the rest of the assumptions which may
 384 still be of the form $\Gamma \leq _$ —which is extremely tedious to do by hand, but easy to automate.

385 3.3 Check-names

386 From the definitions of `pSet` and `bSet`, one immediately sees that there is a canonical map
 387 `check : pSet \rightarrow bSet \mathbb{B}` , defined by

```

388 def check : pSet  $\rightarrow$  bSet  $\mathbb{B}$ 
389 |  $\langle \alpha, A \rangle := \langle \alpha, \lambda a, \text{check } (A a), \lambda a, \top \rangle$ 
390
391

```

392 That is, `check` takes a `pSet` and recursively attaches the Boolean truth-value \top to all
 393 elements. We call members of the image of `check` *check-names*,⁴ after the usual diacritic
 394 notation \check{x} for `check (x : pSet)`. These are also known as *canonical names*, as they are
 395 the canonical representation of standard two-valued sets inside a Boolean-valued model of
 396 set theory. We were pleased to discover Lean’s support for custom notation allowed us to
 397 declare the Unicode modifier character U+030C = “˘” as a postfix operator for `check`.

398 3.4 The fundamental theorem of forcing

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

⁴ This terminology is standard, c.f. [?, ?].

404 Bell [?] gives an extremely detailed account of the verification of the ZFC axioms, and we
 405 faithfully followed his presentation for this part of the formalization. Most of it is routine.
 406 We describe some aspects of `bSet` \mathbb{B} which are revealed by this verification.

407 **The axiom of infinity** $\omega : \text{bSet } \mathbb{B}$ is $\check{\omega}$. ω is defined in `pSet` to be the collection of all
 408 finite von Neumann ordinals, which are defined by induction on \mathbb{N} . While it is easy to show
 409 $\check{\omega}$ satisfies the axiom of infinity

```
410
411 def axiom_of_infinity_spec (u : bSet  $\mathbb{B}$ ) :  $\mathbb{B}$  :=
412   ( $\emptyset \in^{\mathbb{B}} u$ )  $\sqcap$  ( $\bigcap i\_x, \bigcup i\_y, (u.\text{func } i\_x \in^{\mathbb{B}} u.\text{func } i\_y)$ )
413
```

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

416 **The axiom of powerset**

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

```
419
420 def set_of_indicator {x} ( $\chi : x.\text{type} \rightarrow \mathbb{B}$ ) :=  $\langle x.\text{type}, x.\text{func}, \chi \rangle$ 
421
```

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

```
424
425 def bv_powerset (u : bSet  $\mathbb{B}$ ) : bSet  $\mathbb{B}$  :=
426    $\langle u.\text{type} \rightarrow \mathbb{B}, \lambda f, \text{set\_of\_indicator } f, \lambda f, \text{set\_of\_indicator } f \subseteq^{\mathbb{B}} u \rangle$ 
427
```

428 **The axiom of choice** Following Bell, we verified Zorn’s lemma, which is provably equivalent
 429 over ZF to the axiom of choice. As is the case with `pSet`, establishing the axiom of choice
 430 requires the use of a choice principle from the metatheory. This was the most involved part
 431 of our verification of the fundamental theorem of forcing, and relies on the technical tool
 432 of *mixtures*, which allow sequences of \mathbb{B} -valued sets to be “averaged” into new ones, and
 433 the *maximum principle*, which allows existentially quantified statements to be instantiated
 434 without changing their truth-value.

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

440 4 Forcing

441 4.1 Representing Lean’s ordinals inside `pSet` and `bSet`

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

```

447
448 def ordinal.mk : ordinal → pSet
449 | 0 := ∅
450 | succ ξ := pSet.succ (ordinal.mk ξ) -- (mk ξ ∪ {mk ξ})
451 | is_limit ξ := ⋃ η < ξ, (ordinal.mk η)
452

```

Composing by `check` (??) yields a map `check ∘ ordinal.mk : ordinal → bSet \mathbb{B}` . (We could just as well defined `ordinal.mk' : ordinal → bSet \mathbb{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 in Lean are defined separately from ordinals as bijective equivalence classes of types, but are canonically represented by ordinals which 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. “ \aleph_2^\sim ”) 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 $2^{\aleph_2 \times \mathbb{N}}$. The poset of forcing condition typically used for Cohen forcing embeds in this Boolean algebra as a dense suborder.

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

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

```

471
472
473 structure C : Type :=
474   (ins : finset (ℵ2.type × ℕ))
475   (out : finset (ℵ2.type × ℕ))
476   (H : ins ∩ out = ∅)
477

```

That is, we identify a finite partial function f with the triple $\langle f.ins, f.out, f.H \rangle$, 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 $RO(X)$, 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 $\neg CH$ is $RO(2^{\aleph_2 \times \mathbb{N}})$. Unless stated otherwise, for the rest of this section, we put $\mathbb{B} := RO(2^{\aleph_2 \times \mathbb{N}})$.

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

492
493
494

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

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

497
498
499

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

500 Recalling that \leq in B is subset-inclusion, we see that this is essentially because the image of
501 $\iota : C \rightarrow B$ is the standard basis for the product topology. Our chosen encoding of the Cohen
502 poset also made it easier to perform this identification when formalizing this proof.

503 4.3 Adding \aleph_2 -many distinct Cohen reals

504 As we saw in Definition 4, for any B -valued set x , characteristic functions into B from the
505 underlying type of x determine B -valued subsets of x . While the ingredients \aleph_2 and \mathbb{N} for
506 B are types and thus external to $\mathbf{bSet} \ B$, they are represented nonetheless inside $\mathbf{bSet} \ B$
507 by their check-names $\check{\aleph}_2$ and $\check{\mathbb{N}}$, and in fact \aleph_2 is $\check{\aleph}_2$.type and \mathbb{N} is $\check{\mathbb{N}}$.type. Given our
508 specific choice of B , this will allow us to construct an \aleph_2 -indexed family of distinct subsets of
509 $\check{\mathbb{N}}$, which we can then convert into an injective function from $\check{\aleph}_2$ to $\check{\mathbb{N}}$, inside $\mathbf{bSet} \ B$.

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

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

515 This gives us an \aleph_2 -indexed family of Cohen reals. Converting this data into an injective
516 function from $\check{\aleph}_2$ to $\check{\mathbb{N}}$ inside $\mathbf{bSet} \ B$ requires some care. One must check that $\nu \mapsto \widetilde{\chi}_\nu$ is
517 externally injective, and this is where the characterization of the Cohen poset as a dense
518 subset of B (and moving back and forth between this representation and the definition as
519 finite partial functions) comes in. Furthermore, one has to develop machinery similar to that
520 for the powerset operation to convert an external injective function $\mathbf{x.type} \rightarrow \mathbf{bSet} \ B$ to a
521 B -valued set which $\mathbf{bSet} \ B$ thinks is a injective function, while maintaining conditions on
522 the intended codomain. Our custom tactics and automation for reasoning inside B made this
523 latter task significantly easier than it would have been otherwise. We refer the interested
524 reader to our formalization for details.

525 4.4 Preservation of cardinal inequalities

526 So far, we have shown for $B = \mathbf{RO}(2^{\aleph_2 \times \mathbb{N}})$ that $\mathbf{bSet} \ B$ thinks $\check{\aleph}_2$ is smaller than $\mathcal{P}(\check{\mathbb{N}})$.
527 Although Lean believes there is a strict inequality of cardinals $\aleph_0 < \aleph_1 < \aleph_2$, in general
528 we can only deduce that their representations inside $\mathbf{bSet} \ B$ are subsets of each other:
529 $\top \leq \check{\aleph}_0 \subseteq^B \check{\aleph}_1 \subseteq^B \check{\aleph}_2$. To finish negating CH, it suffices to show that $\mathbf{bSet} \ B$ thinks $\check{\aleph}_0$ is
530 strictly smaller than $\check{\aleph}_1$, and that $\mathbf{bSet} \ B$ thinks $\check{\aleph}_1$ is a strictly smaller than $\check{\aleph}_2$. That is,
531 we want that the passage from \aleph_i to $\check{\aleph}_i$ preserves cardinal inequalities.

532 ► **Definition 10.** For our purposes, “ X is strictly smaller than Y ” means “there exists no
533 function \mathbf{f} such that for every $\mathbf{y} \in Y$, there exists an $\mathbf{x} \in X$ such that $(\mathbf{x}, \mathbf{y}) \in \mathbf{f}$ ”. Thus, “ X
534 is strictly smaller than Y ” translates to the Boolean truth-value

535 $\neg(\bigsqcup f, (\text{is_func } f) \sqcap \bigsqcup y, y \in^{\mathbb{B}} Y \implies \bigsqcup x, x \in^{\mathbb{B}} X \sqcap (x, y) \in^{\mathbb{B}} f).$

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

538 ▶ **Definition 11.** We say that \mathbb{B} has the **countable chain condition** (CCC) if every
539 antichain $\mathcal{A} : I \rightarrow \mathbb{B}$ (i.e. an indexed collection of elements $\mathcal{A} := \{a_i\}$ such that whenever
540 $i \neq j, a_i \sqcap a_j = \perp$) has a countable image.

541 We sketch the argument that CCC implies the preservation of cardinal inequalities. The
542 proof is by contraposition. Let κ_1 and κ_2 be cardinals such that $\kappa_1 < \kappa_2$, and suppose that
543 $\check{\kappa}_1$ is not strictly smaller than $\check{\kappa}_2$. Then there exists some $f : \mathbf{bSet } \mathbb{B}$ and some $\Gamma > \perp$ such
544 that $\Gamma \leq (\text{is_func } f) \sqcap \bigsqcup y, y \in^{\mathbb{B}} \check{\kappa}_1 \implies \bigsqcup x, x \in^{\mathbb{B}} \check{\kappa}_2 \sqcap (x, y) \in^{\mathbb{B}} f$. Then one
545 can show:

546 **lemma** `AE_of_check_larger_than_check` :
547 $\forall \beta < \kappa_2, \exists \eta < \kappa_1, \perp < (\text{is_func } f) \sqcap (\check{\eta}, \beta^\sim) \in^{\mathbb{B}} f$
548
549

550 The name of this lemma emphasizes that what was happened here is that, given this f and
551 the assumption that it satisfies some $\forall\text{-}\exists$ formula inside $\mathbf{bSet } \mathbb{B}$, we are able to extract, by
552 virtue of $\check{\kappa}_1$ and $\check{\kappa}_2$ being check-names, a $\forall\text{-}\exists$ statement in the *metatheory*. Using Lean's
553 choice principle, we can then convert this $\forall\text{-}\exists$ statement into a function $g : \kappa_2 \rightarrow \kappa_1$, such
554 that for every $\beta, \perp < (\text{is_func } f) \sqcap (g(\beta)^\sim, \beta^\sim) \in^{\mathbb{B}} f$. Since $\kappa_2 > \kappa_1$, it follows from
555 the infinite pigeonhole principle that there exists some $\eta < \kappa_1$ such that the $g^{-1}(\{\eta\})$ is
556 uncountable. Define $\mathcal{A} : g^{-1}(\{\eta\}) \rightarrow \mathbb{B}$ by $\mathcal{A}(\beta) := (\text{is_func } f) \sqcap (g(\beta)^\sim, \beta^\sim) \in^{\mathbb{B}} f$.
557 This is an uncountable antichain because if $\beta_1 \neq \beta_2$, then the well-definedness part of
558 $\text{is_func } f$ ensures that, since $g(\beta_1) = g(\beta_2)$, the truth-value $\beta_1^\sim = f(g(\beta_1)) \neq^{\mathbb{B}} f(g(\beta_2)) =$
559 β_2^\sim is \perp .

560 Thus, conditional on showing that $\mathbb{B} = \text{RO}(2^{\aleph_2 \times \aleph})$ has the CCC, we now have that
561 cardinal inequalities are preserved in $\mathbf{bSet } \mathbb{B}$. Combining this with the injection $\aleph_2^\sim \rightarrow \mathcal{P}$
562 (\aleph) , we obtain:

563 **theorem** `neg_CH` : $\top = (\aleph < (\aleph_1)^\sim \sqcap (\aleph_1)^\sim < (\aleph_2)^\sim \sqcap (\aleph_2)^\sim \leq \mathcal{P}(\aleph))$
564
565

566 The arguments sketched in subsection 4.3 and subsection 4.4 form the heart of the
567 forcing argument. Their proofs involve taking objects in `Type` u and $\mathbf{bSet } \mathbb{B}$, constructing
568 corresponding objects on the other side, and reasoning about them in ordinary and \mathbb{B} -valued
569 logic simultaneously to determine cardinalities in $\mathbf{bSet } \mathbb{B}$. We have omitted many details
570 from our discussion, but of course, all the proofs have been formally verified.

571 4.5 The unprovability of CH

572 We conclude this section by briefly describing how the previous results may be converted
573 into a formal proof of the unprovability of CH. We work in a conservative expansion ZFC' of
574 ZFC with an expanded language $L_{\text{ZFC}'}$ with symbols for pairing, union, powerset, and ω . We
575 define ZFC' to be precisely the ZFC axioms which were verified in the fundamental theorem
576 of forcing, along with specifications for the new function symbols. CH can then be written as
577 a deeply-embedded $L_{\text{ZFC}'}$ sentence

578 **def** `CH` : `sentence` $L_{\text{ZFC}'}$:= $\neg \exists' \exists' (\omega < \&1) \sqcap (\&1 < \&0) \sqcap (\&0 \leq \mathcal{P}(\omega))$
579
580

581 where $<$ and \leq are abbreviations with the same meaning as in the previous section. Then
582 proving $\mathbf{bSet } \mathbb{B} \models \text{ZFC}' + \neg \text{CH}$ is a straightforward matter of checking that sentences are

583 interpreted correctly as Boolean truth values which we have already proved to be \top . Applying
 584 the contrapositive of the Boolean-valued soundness theorem yields the result.

585 **5 Transfinite combinatorics and the countable chain condition**

586 What remains now is to prove that $\text{RO}(2^{\aleph_2 \times \aleph})$ has the CCC. There are several ways forward;
 587 we chose the most general one, anticipating its usefulness in future formalizations of set
 588 theory and forcing.

589 **5.1 The Δ -system lemma**

590 **► Definition 12.** A Δ -system is... Lorem ipsum dolor sit amet, consectetur adipiscing elit.
 591 Donec molestie rutrum sapien et sagittis. Curabitur varius egestas tortor. Sed faucibus
 592 tincidunt felis, et tincidunt erat consequat eu. Praesent ullamcorper interdum ex in ornare.
 593 Vestibulum quam sem, molestie ac aliquam sit amet, efficitur non nunc. Suspendisse rutrum
 594 metus vitae ligula sagittis.

```
595 def is_delta_system {α : Type u} (A : set (set α)) :=
596   ∃(root : set α), ∀{x y}, x ∈ A → y ∈ A → x ≠ y → x ∩ y = root
597
598
```

599 TODO(floris) Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec molestie
 600 rutrum sapien et sagittis. Curabitur varius egestas tortor. Sed faucibus tincidunt felis,
 601 et tincidunt erat consequat eu. Praesent ullamcorper interdum ex in ornare. Vestibulum
 602 quam sem, molestie ac aliquam sit amet, efficitur non nunc. Suspendisse rutrum metus
 603 vitae ligula sagittis, commodo lacinia metus ultricies. Sed ultricies fringilla magna ac luctus.
 604 Vivamus dolor mauris, vulputate in tempus dictum, faucibus non eros. Praesent aliquet,
 605 justo et tristique interdum, tellus sapien tempus sem, eu vestibulum nisl sem at ligula.
 606 Aenean commodo mauris nec leo pellentesque porttitor. Class aptent taciti sociosqu ad litora
 607 torquent per conubia nostra, per inceptos himenaeos. Duis interdum eget nulla ac molestie.
 608 Proin vel sem auctor, porttitor velit nec, cursus arcu.

```
609
610
611 theorem delta_system_lemma {α : Type u} {κ : cardinal}
612   (hκ : cardinal.omega ≤ κ) {θ} (hκθ : κ < θ) (hθ : is_regular θ)
613   (hθ_le : ∀(c < θ), c < κ < θ) (A : set (set α))
614   (hA : θ ≤ mk A) (h2A : ∀{s : set α} (h : s ∈ A), mk s < κ) :
615     ∃(B ⊆ A), mk B = θ ∧ is_delta_system B :=
616
617
```

618 Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec molestie rutrum sapien
 619 et sagittis. Curabitur varius egestas tortor. Sed faucibus tincidunt felis, et tincidunt erat
 620 consequat eu. Praesent ullamcorper interdum ex in ornare. Vestibulum quam sem, molestie
 621 ac aliquam sit amet, efficitur non nunc. Suspendisse rutrum metus vitae ligula sagittis,
 622 commodo lacinia metus ultricies. Sed ultricies fringilla magna ac luctus. Vivamus dolor
 623 mauris, vulputate in tempus dictum, faucibus non eros. Praesent aliquet, justo et tristique
 624 interdum, tellus sapien tempus sem, eu vestibulum nisl sem at ligula. Aenean commodo
 625 mauris nec leo pellentesque porttitor.

5.2 $\text{RO}(2^{\mathbb{N}_2 \times \mathbb{N}})$ has the countable chain condition

TODO(floris) Lorem ipsum dolor sit amet, consectetur adipiscing elit. Donec molestie rutrum sapien et sagittis. Curabitur varius egestas tortor. Sed faucibus tincidunt felis, et tincidunt erat consequat eu. Praesent ullamcorper interdum ex in ornare. Vestibulum quam sem, molestie ac aliquam sit amet, efficitur non nunc. Suspendisse rutrum metus vitae ligula sagittis, commodo lacinia metus ultricies. Sed ultricies fringilla magna ac luctus. Vivamus dolor mauris, vulputate in tempus dictum, faucibus non eros. Praesent aliquet, justo et tristique interdum, tellus sapien tempus sem, eu vestibulum nisl sem at ligula. Aenean commodo mauris nec leo pellentesque porttitor. Class aptent taciti sociosqu ad litora torquent per conubia nostra, per inceptos himenaeos. Duis interdum eget nulla ac molestie. Proin vel sem auctor, porttitor velit nec, cursus arcu.

```
theorem B_CCC : CCC regular_open_algebra :=
begin
  simp[Suspendisse rutrum metus vitae ligula sagittis,
        commodo lacinia metus ultricies,
        Sed ultricies fringilla magna ac luctus,
        Vivamus dolor mauris, vulputate in tempus dictum];
  tidy
end
```

6 Related work

First-order logic, soundness, and completeness There are many existing formalizations of first-order logic. Shankar [?] used a deep embedding of first-order logic to formalize incompleteness theorems. Harrison gives a deeply-embedded implementation of first-order logic in HOL Light [?] and a proof-search style account of the completeness theorem in [?]. Margetson [?] and Schlichtkrull [?] use the same argument for the completeness theorem in Isabelle/HOL, while Berghofer [?] (in Isabelle) and Ilik [?] (in Coq) use canonical term models.

Set theory and forcing A large body of formalized set theory has been completed in Isabelle/ZF, led by Paulson and his collaborators [?, ?, ?]. Most relevantly, this includes a formalization of the relative consistency of the axiom of choice with ZF [?]. Building on this work, Gunther, Pagano, and Terraf have begun formalizing the basic ingredients of forcing [?, ?].

Our embedded proof language for Boolean-valued logic was partly inspired by similar work in the `unitb` project⁵. It was pointed out to the authors that a trick similar to Lemma 3 has been successfully applied in the Metamath library [?]

The work we have described in this paper relies heavily on Lean’s `mathlib`. In particular, the extensive `set_theory` and `ordinal` libraries contained nearly everything we needed (including a treatment of cofinalities for the Δ -system lemma), and what was missing was accessible by combining a few existing lemmas. These libraries were originally developed by

⁵ <https://github.com/unitb/temporal-logic>

668 Carneiro [?], in part to show that Lean proves the existence of infinitely many inaccessible
 669 cardinals.

670 7 Conclusions and future work

671 **Reflections on the proof** As our formalization has shown, for the purposes of a consistency
 672 proof, one can perform forcing entirely outside of the set-theoretic foundations in which
 673 forcing is usually presented. There is no need to work inside an ambient model of set theory,
 674 or to even have a ground model of set theory over which one constructs a forcing extension.
 675 Instead, the recursive *name* construction (generalizing the Aczel-Werner encoding) applied to
 676 a universe of types is the key. The type universe, with its classical two-valued metatheory and
 677 its own notion of ordinals, takes the place of the standard universe of sets. These external
 678 ordinals are then represented in the internal ordinals of the forcing extension by indexing the
 679 construction of von Neumann ordinals. With a clever choice of forcing conditions \mathbb{B} , one can
 680 make this representation of ordinals send externally distinct cardinals to internally distinct
 681 cardinals, and then force an uncountable cardinal beneath $\mathcal{P}(\mathbb{N})$.

682 In particular, `pSet`, being only another special case of the construction which produces
 683 `bSet` \mathbb{B} , is no longer a prerequisite for working with `bSet` \mathbb{B} , but merely a convenient tool
 684 for organizing the check-names—this is the only role it played in the proof. The check-
 685 names themselves were actually not necessary either: as we remarked, the canonical map
 686 `ordinal` \rightarrow `bSet` \mathbb{B} can be defined without reference to them. However, since in all of our
 687 sources, `pSet` additionally played the role of the universe of types, and an interface for it was
 688 readily available in `mathlib`, we started our formalization by following the usual arguments,
 689 implementing these simplifications as we became aware of them.

690 Lessons learned

- 691 ■ Originally, we thought set-theoretic arguments involving transfinite/ordinal induction,
 692 which are ubiquitous, would be difficult to implement. In practice, Lean’s tools for
 693 well-founded recursion and the comprehensive treatment of ordinals in `mathlib` made
 694 the implementation of such arguments painless.
- 695 ■ Definitions and lemmas should be stated as generally as possible. This maximizes
 696 reusability, minimizes redundancy, and by exposing only the information required to
 697 complete the proof, improves the performance of automation.
- 698 ■ One should invest early in domain-specific automation. The formalization of the funda-
 699 mental theorem was completed using only the first two strategies outlined in subsection 3.2;
 700 the calculations, while tedious, were recorded in our sources and it seemed easier to follow
 701 them. If we had followed through on the observations around Lemma 3 and developed
 702 the custom tactic library earlier, we would have saved a significant amount of time.

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

704 The work we have described in this paper was undertaken as part of the Flypitch project,
 705 which aims to produce a formal proof of the independence of the continuum hypothesis. As
 706 such, the obvious next goal is a formalization of the consistency of the continuum hypothesis.

707 As indicated at the start of section 1, a formal proof of the independence of CH means
 708 showing that CH and \neg CH cannot be proved from the ZFC axioms. Although our work
 709 includes a formal proof of the unprovability of a version of CH from a version of the ZFC
 710 axioms in a conservative extension of the language of ZFC, verifying this is easy.

711 What is more interesting is formalizing the equivalence of various common formulations
712 of ZFC and CH, so that a skeptical user may verify that their preferred version of CH is
713 unprovable from their preferred version of ZFC. This would require formalizations of the
714 conservativity of commonly-used extensions of ZFC, and of the equivalence of the various
715 ways to say that one set is strictly smaller than another.

716 Although the stated goal of our project is to achieve a formal proof of the independence
717 of the continuum hypothesis, we also intend to develop reusable libraries for set theory and
718 mathematical logic. We have completed a formalization of forcing, but are nowhere near
719 completing a library which a set theorist could use to verify their research. Just as, more
720 than 50 years ago, Cohen's proof marked the beginning of modern research in set theory, a
721 formal proof of the independence of the continuum hypothesis will only mark the beginning
722 of an integration of formal methods into modern research in set theory. This will require
723 robust interfaces for handling the diverse range of forcing arguments and for reasoning about
724 the consistency strengths of various extensions of ZFC, so that—to paraphrase Kanamori [?]
725 deeply-embedded notions of truth and relative consistency become matters of manipulation
726 as in algebra. Our work demonstrates that such tasks are well within the scope of modern
727 interactive theorem provers.

728 8 References

729 TODO