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

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

Floris van Doorn
Department of Mathematics, University of Pittsburgh

Abstract

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

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

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

Digital Object Identifier 10.4230/LIPIcs.CVIT.2016.23

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

Introduction

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

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

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

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

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

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

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

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

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, Boolean-valued semantics, 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 $\mathbb{B}$ for forcing $\neg$CH. Section 5 describes the formalization of the Δ-system lemma, which we use to prove forcing with $\mathbb{B}$ preserves cardinal inequalities. We conclude with a reflection on 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 ZFC $\vdash$ CH, then CH interpreted in any model of ZFC is true. Taking the contrapositive, we can demonstrate the unprovability (equivalently, the consistency of the negation) of CH by exhibiting a single model where CH is not true.

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

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

To force the negation of the continuum hypothesis, we use the Boolean algebra of regular opens of the Cantor space $\mathbb{B} := \mathrm{RO}(2^{\aleph_2 \times \mathbb{N}})$. For each $\nu \in \aleph_2$, we associate the $\mathbb{B}$-valued characteristic function $\chi_\nu : \mathbb{N} \to \mathbb{B}$ by $n \mapsto \{f \mid f(\nu, n) = 1\}$. This induces what $V^{\mathbb{B}}$ thinks is a new subset $\widetilde{\chi_\nu} \subseteq \mathbb{N}$, called a *Cohen real*, and furthermore, simultaneously performing this construction on all $\nu : \aleph_2$ induces what $V^{\mathbb{B}}$ thinks is a function from $\check{\aleph}_2 \to \mathcal{P}(\mathbb{N})$. After showing that $V^{\mathbb{B}}$ thinks this function is injective, to finish the proof it suffices to show that $x \mapsto \check{x}$ preserves cardinal inequalities, as then we will have squeezed $\check{\aleph}_1$ properly between $\mathbb{N}$ and $\mathcal{P}(\mathbb{N})$. This is really the technical heart of the matter, and 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 $2^{\aleph_2 \times \mathbb{N}}$, which we handle 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 ofZFCsatisfying 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 should be read as Lean-like pseudocode. For the sake of formatting and readability, names, universe levels, type ascriptions, and casts have been removed or changed. We refer the interested reader to our source code[3]. The forcing argument for the negation of CH is located in `forcing.lean`. In a Lean-aware editor such as Emacs, the user is encouraged start at the theorem `neg_CH` and jump backwards to trace the dependencies of the proof. (TODO(jesse) maybe change this to point at the "summary" file)

2 First-order logic

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

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

2.1 (Pre)terms, (pre)formulas

The main novelty of our implemenation 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 : ℕ → Type u
| var {} : ∀ (k : ℕ), preterm 0
| func : ∀ {l : ℕ} (f : L.functions l), preterm l
```

[3] `https://github.com/flypitch/flypitch`

```
| app : ∀ {l : ℕ} (t : preterm (l + 1)) (s : preterm 0), preterm l
```

We use de Bruijn indices to avoid variable shadowing. A member of `preterm L 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 : ℕ → Type u
| falsum {} : preformula 0
| equal (t₁ t₂ : term L) : preformula 0
| rel {l : ℕ} (R : L.relations l) : preformula l
| apprel {l : ℕ} (f : preformula (l + 1)) (t : term L) : preformula l
| imp (f₁ f₂ : preformula 0) : preformula 0
| all (f : preformula 0) : preformula 0
```

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

Our proof system is a natural deduction calculus. This makes a proof of the soundness theorem by structural induction easier. All rules are motivated to work well with backwards-reasoning.

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

2.2 Completeness

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

For our formalization, we chose the Henkin-style approach of constructing a canonical term model. In order to perform the argument, which normally involves modifying the language "in place" to iteratively add new constant symbols, we had to adapt it to type theory. Since our languages are represented by pairs of indexed types instead of sets, we cannot really modify them in-place with new constant symbols. Instead, at each step of the construction, we must construct an entirely new language in which the previous one embeds, and in the limit we must compute a directed colimit of types instead of a union.

This construction induces similar constructions on terms and formulas, and completing the argument requires reasoning with all of them. As a result of our design decisions, only a few arguments required anything more than straightforward case-analysis and structural induction. The final statement makes no restrictions on the cardinality of the language.

2.3 Boolean-valued semantics for first-order logic

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

▸ **Definition 1.** Fix a language L and a type β with the structure of a complete Boolean algebra. A **β-valued structure** is an instance of the following `structure`:

```
structure bStructure :=
(carrier : Type u)
(fun_map : ∀{n}, L.functions n → dvector carrier n → carrier)
(rel_map : ∀{n}, L.relations n → dvector carrier n → β)
(eq : carrier → carrier → β)
(eq_refl : ∀ x, eq x x = ⊤)
(eq_symm : ∀ x y, eq x y = eq y x)
(eq_trans : ∀{x} y {z}, eq x y ⊓ eq y z ≤ eq x z)
(fun_congr : ∀{n} (f : L.functions n) (x y : dvector carrier n),
(x.map2 eq y).fInf ≤ eq (fun_map f x) (fun_map f y))
(rel_congr : ∀{n} (R : L.relations n) (x y : dvector carrier n),
  (x.map2 eq y).fInf ⊓ rel_map R x ≤ rel_map R y)
```

Note that Boolean-valued equality is not really an equivalence relation, but "β 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 is 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 {Γ : set (formula L)} {A : formula L} (H : Γ
    ⊢ A) : ∀ M, (⨅γ : Γ, M[γ]) ≤ M[A]
```

We designed our datatype of proofs as an inductive type whose constructors are precisely the natural deduction rules naturally supported by Lean's Prop. As a result, the proofs of both the ordinary and Boolean-valued soundness theorems are straightforward structural inductions.

3 Constructing Boolean-valued models of set theory

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

In set theory (see e.g. Jech [?] or Bell [?]), Boolean-valued models are obtained by imitating the construction of the von Neumann cumulative hierarchy via a transfinite recursion where iterations of the powerset operation (taking functions into $\mathbf{2} = \{\mathrm{true}, \mathrm{false}\}$) are replaced by iterations of the "𝔹-valued powerset operation" (taking functions into 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 encodeZFCinto 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 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 𝔹-names, and will be the underlying type of our Boolean-valued model of set theory. For convenience, if `x : bSet 𝔹` and `x := ⟨α, A, B⟩`, we put `x.type := α, x.func := A, x.bval := B`.

3.1 Boolean-valued equality and membership

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

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

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

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

3.2 Automation and metaprogramming for reasoning in 𝔹

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

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

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

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

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

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

Third, one can make the following observation:

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

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

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

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

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

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

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

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

3.3 Check-names

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

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

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

3.4 The fundamental theorem of forcing

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

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

The axiom of infinity

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

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

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

The axiom of powerset

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

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

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

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

The axiom of choice

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

The smallness of $\mathbb{B}$

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

4 Forcing

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

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

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

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

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

4.2 The Cohen poset and the regular open algebra

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

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

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

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

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

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

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

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

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

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

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

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

4.3 Adding $\aleph_2$-many distinct Cohen reals

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

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

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

This gives us an $\aleph_2$-indexed family of Cohen reals. Converting this data into an injective function from $\aleph_2\check{}$ to $\mathbb{N}$ 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. Furthermore, one has to develop machinery similar to that for the powerset operation to convert an external injective function `x.type → bSet 𝔹` to a $\mathbb{B}$-valued set which `bSet 𝔹` believes is a injective function, while maintaining conditions on the intended codomain. Our custom tactics and automation for reasoning inside $\mathbb{B}$ made this latter task significantly easier than it would have been otherwise. We refer the interested reader to our formalization for details.

4.4 Preservation of cardinal inequalities

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

Although Lean believes there is a strict inequality of cardinals $\aleph_0 < \aleph_1 < \aleph_2$, in general we can only deduce that their representations inside `bSet 𝔹` are subsets of each other: $\top \leqslant \check{\aleph}_0 \subseteq^{\mathbb{B}} \check{\aleph}_1 \subseteq^{\mathbb{B}} \check{\aleph}_2$. To finish negating CH, it suffices to show that `bSet 𝔹` believes $\check{\aleph}_0$ is strictly smaller than $\check{\aleph}_1$, and that `bSet 𝔹` believes $\check{\aleph}_1$ is a strictly smaller than $\check{\aleph}_2$. That is, we want that the passage from $\aleph_i$ to $\check{\aleph}_i$ preserves cardinal inequalities.

▸ **Definition 10.** For our purposes, "strictly smaller" means "there exists no function `f` such that for every `v ∈ y`, there exists a `w ∈ x` such that `(w,v) ∈ f`". With the definition of "is a function" abbreviated, "`x` is strictly smaller than `y`" then translates to the Boolean truth-value

```
    -(⨆f, (is_func f) ⊓ ⨅v, v ∈ᴮ y ⟹ ⨆w, w ∈ᴮ x ⊓ (w, v) ∈ᴮ f).
```

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

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

We sketch the argument that CCC implies the preservation of cardinal inequalities. The proof is by contraposition. Let κ_1 and κ_2 be cardinals such that $\kappa_1 < \kappa_2$, and suppose that $\check{\kappa_1}$ is not strictly smaller than $\check{\kappa_2}$. Then there exists some `f : bSet` $\mathbb{B}$ and some $\Gamma > \bot$ such that Γ `⩽ (is_func f) ⊓ ⨅v, v ∈ᴮ κ₁ˇ ⟹ ⨆w, w ∈ᴮ κ₂ˇ ⊓ (w,v) ∈ᴮ f`. Then one can show:

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

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

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

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

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

5 Transfinite combinatorics and the countable chain condition

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

5.1 The Δ-system lemma

▸ **Definition 12.** A **Δ-system** is... 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.

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

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 delta_system_lemma {α : Type u} {κ : cardinal}
  (hκ : cardinal.omega ⩽ κ) {θ} (hκθ : κ < θ) (hθ : is_regular θ)
  (hθ_le : ∀(c < θ), c ^< κ < θ) (A : set (set α))
  (hA : θ ⩽ mk A) (h2A : ∀{s : set α} (h : s ∈ A), mk s < κ) :
    ∃(B ⊆ A), mk B = θ ∧ is_delta_system B :=
```

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.

5.2 $\mathrm{RO}(2^{\aleph_2 \times \mathbb{N}})$ 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

TODO

7 Conclusions and future work

Reflections on the proof

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

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

Lessons learned

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

Towards a formal proof of the independence of the continuum hypothesis

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

As indicated at the start of section 1, this means a formal proof of the independence of CH means showing that CH and ¬CH cannot be proved from the ZFC axioms. Although our work includes a formal proof of the unprovability of a version of CH from a version of the ZFC axioms in a conservative extension of the language of ZFC, verifying this is only a matter of checking that the deeply-embedded formulas are correctly interpreted as Boolean values which have already been proven equal to $\top$.

What is more interesting is formalizing the equivalence of various common formulations of ZFC and CH, so that a skeptical user may verify that their preferred version of CH is unprovable from their preferred version of ZFC. This would require formalizations of the conservativity of commonly-used extensions of ZFC, and of the equivalence of the various ways to say that one set is strictly smaller than another. The completeness theorem will be useful for this, because it constructs deeply-embedded proofs from ordinary proofs carried out in an arbitrary model.

Combined with the completeness theorem, the Boolean-valued soundness theorem should allow us to show arbitrary theorems of ZFC are true in `bSet` $\mathbb{B}$ by carrying out the proofs in an arbitrary two-valued model, thus avoiding working directly inside Boolean-valued logic. In this way, we can transport theorems such as "Zorn's lemma is equivalent to the axiom of choice" directly from the world of two-valued models to the world of Boolean-valued models.

We also intend to automate reflective procedures for recognizing when an expression in a deeply-embedded model is an instance for a formula. For example, the "real" proof that something like the subset predicate is $=^B$-extensional is a proof by reflection: one constructs a formula which reifies the predicate, and then applies the fact that one is in the deeply-embedded Boolean-valued structure to obtain the congruence lemma automatically.

8 References

TODO