A Henkin-Style Completeness Proof for the Modal Logic S5

Bruno Bentzen(✉)

Carnegie Mellon University, Pittsburgh, PA, USA

Abstract. This paper presents a recent formalization of a Henkin-style completeness proof for the propositional modal logic S5 using the Lean theorem prover. The proof formalized is close to that of Hughes and Cresswell [8], but the system, based on a different choice of axioms, is better described as a Mendelson system augmented with axiom schemes for K, T, S4, and B, and the necessitation rule as a rule of inference. The language has the false and implication as the only primitive logical connectives and necessity as the only primitive modal operator. The full source code is available online and has been typechecked with Lean 3.4.2.

Keywords: Modal logic · Completeness · Formal methods · Lean

1 Introduction

A proof of the completeness theorem for a given logic conforms to the Henkin style when it applies nonconstructive methods to build models out of maximal consistent sets of formulas (possibly after a Henkin language extension) using the deductive system itself. Henkin-style completeness proofs for modal logics have been around for over five decades [9] but the formal verification of completeness with respect to Kripke semantics is comparatively recent.

We present a formalization of a Henkin-style completeness proof for the propositional modal logic S5 using the Lean theorem prover. Although the proof is specific to S5, the same techniques can be applied to weaker normal modal systems such as K, T, S4, and B, by forgetting the appropriate extra accessibility conditions (as described in [8]). The formalization covers the syntax and semantics of S5, the deduction theorem, structural rules (weakening, contraction, exchange), the recursive enumerability of the language, and the soundness and completeness theorems. It has approximately 1,500 lines of code (but only two thirds of the development is required for the completeness proof). The full source code is available online. It has been typechecked with Lean 3.4.2.

The author thanks Jeremy Avigad, Mario Carneiro, Rajeev Goré, and Minchao Wu for helpful suggestions. An early version of this work was presented at the Lean Together 2019, Amsterdam, January 7–11, 2019. The source code described in this paper is publicly available online at: https://github.com/bbentzen/mpl/. An extended version is available at https://arxiv.org/abs/1910.01697.

P. Baroni et al. (Eds.): CLAR 2021, LNAI 13040, pp. 459–467, 2021.
https://doi.org/10.1007/978-3-030-89391-0_25

1.1 Related Work

The use of proof assistants in the mechanization of completeness proofs in the context of Kripke semantics has been recently studied in the literature for a variety of formal systems. Coquand [2] uses ALF to give a constructive formal proof of soundness and completeness w.r.t. Kripke models for intuitionistic propositional logic with implication as the sole logical constant. Building on Coquand's work, a constructive completeness proof w.r.t Kripke semantics for intuitionistic logic with implication and universal quantification has been verified with Coq by Heberlin and Lee [7]. Also using Coq, Doczal and Smolka present a constructive formal proof of completeness w.r.t. Kripke semantics and decidability of the forcing relation for an extension of modal logic K [4] and a variety of temporal logic [5]. In his formal verification of cut elimination for coalgebraic logics, Tews [13] provides a formalization of soundness and completeness proofs covering many different logics, including modal logic K.

To the best of our knowledge, our formalization of a Henkin-style completeness proof for propositional modal logic S5 is the first of its kind.[1] Our proof is close to that of Hughes and Cresswell [8], but given for a system based on a different choice of axioms. In Hughes and Cresswell's book, the basis of S5 is that of T plus an additional axiom. Here S5 is built on axiom schemes for K, T, S4 and B. This has the advantage that we can easily adapt the proof for different weaker systems. Another choice had to be made between using a deep or a shallow embedding for the formalization. Because our aim is metatheoretical, we use a deep embedding for the encoding of the syntax, as it allows us to prove metatheorems by structural induction on formulas or derivations.

1.2 Lean

Lean is an interactive theorem prover developed principally by Leonardo de Moura based on a version of dependent type theory known as the Calculus of Inductive Constructions [3,11]. Theorem proving in Lean can be done by constructing proof terms directly, as in Agda [10], by using tactics, imperative commands that describe how to construct a proof term, as in Coq [12], or by mixing them together in the same environment. Lean also supports classical reasoning, which is employed in the formalization along with the declaration of noncomputable constructions. The formalization also presupposes a few basic results on data structures which are not in the standard library, so, our development makes use of `mathlib`, the library of formalized mathematics for Lean [1].

In the remainder of this paper, Lean code will be used to illustrate design choices and give an overview of the proof method, not to discuss the proof itself. Interested readers are encouraged to consult `completeness.lean`, the main file

[1] In independent work done roughly at the same time the author first completed this formalization in 2018, Wu and Goré [14] have described a formalization in Lean of modal tableaux for modal logics K, KT, and S4 with decision procedures with proofs of soundness and completeness. Also in 2018, but unknown to the author, From [6] formalized a Henkin-style completeness proof for system K in Isabelle.

of the formalization where the crucial proof steps are given in detail. We shall also give an informal proof sketch of the completeness theorem using mathematical notation to convey the key ideas of the proof.

2 Modal Logic

2.1 The Language

For simplicity, we shall work with a language which has implication ($\supset$) and the false ($\bot$) as the only primitive logical connectives, and necessity ($\Box$) as the only primitive modal operator. This language can be conveniently defined using inductive types in Lean:

```
inductive form {σ :nat} : Type
| atom : fin σ → form
| bot  : form
| impl : form → form → form
| box  : form → form
```

Using one of the four constructors displayed above (`atom`, `bot`, `impl`, `box`) is the only way to construct a term of type `form`. The elimination rule of this type is precisely the principle of induction on the structure of a formula.

While newly-defined terms are always exhibited in Polish notation by default, Lean supports unicode characters and has an extensible parser which allows the declaration of customized prefix or infix notations for terms and types.

```
prefix    '#'       := form.atom
notation '⊥'       := form.bot _
infix     '⊃'       := form.impl
notation '∼':40 p  := form.impl p (form.bot _)
prefix    '□':80    := form.box
prefix    '◊':80    := λ p, ∼(□ (∼ p))
```

Contexts are just sets of formulas. In Lean, sets are functions of type `A → Prop`:

```
def ctx : Type := set (form σ)
notation '·' := {}
notation Γ ',' p :=set.insert p Γ
```

2.2 The Proof System

We define a Mendelson system augmented with axiom schemes for K, T, S4, and B, and the necessitation rule as a rule of inference. The proof system is implemented with a type of proofs, which is inductively defined as follows:

```
inductive prf : ctx σ → form σ → Prop
| ax {Γ} {p} (h :p∈ Γ): prf Γ p
| pl1 {Γ} {p q}       : prf Γ (p ⊃ (q ⊃ p))
| pl2 {Γ} {p q r}     : prf Γ ((p ⊃ (q ⊃ r)) ⊃ ((p ⊃ q) ⊃ (
   p ⊃ r)))
```

```
| pl3 {Γ} {p q}        : prf Γ (((∼p) ⊃ ∼q) ⊃ (((∼p) ⊃ q) ⊃
    p)
| k {Γ} {p q}          : prf Γ (□(p ⊃ q) ⊃ (□p ⊃ □q))
| t  {Γ} {p}           : prf Γ (□p ⊃ p)
| s4 {Γ} {p}           : prf Γ (□p ⊃ □□p)
| b {Γ} {p}            : prf Γ (p ⊃ □◊p)
| mp {Γ} {p q} (hpq: prf Γ (p ⊃ q)) (hp :prf Γ p) :prf Γ q
| nec {Γ} {p} (h :prf · p) : prf Γ (□p)

notation Γ ‘ ⊢_s5 ’ p :=prf Γ p
notation Γ ‘ ⊬_s5 ’ p :=prf Γ p → false
```

2.3 Semantics

Kripke Models. The semantics for S5 are given by Kripke semantics. A model $\mathcal{M}$ is a triple $\langle \mathcal{W}, \mathcal{R}, v \rangle$ where

- $\mathcal{W}$ is a non-empty set of objects called possible worlds;
- $\mathcal{R}$ is a binary, equivalence relation on possible worlds;
- v specifies the truth value of a formula at a world.

It is useful to let the members of $\mathcal{W}$ (possible worlds) be sets of formulas:

```
def wrld (σ :nat) := set (form σ)
```

Kripke models can be implemented as structures (inductive types with only one constructor). This can be done using the `structure` command in Lean. We define a 6-tuple composed of a domain, an accessibility relation, a valuation function, and reflexivity, symmetry and transitivity proofs for the given relation:

```
structure model :=
(wrlds : set (wrld σ))
(access : wrld σ → wrld σ → bool)
(val : fin σ → wrld σ → bool)
(refl : ∀ w ∈ wrlds, access w w =tt)
(symm : ∀ w ∈ wrlds, ∀ v ∈ wrlds, access w v =tt → access v
    w =tt)
(trans : ∀ w ∈ wrlds, ∀ v ∈ wrlds, ∀ u ∈ wrlds,
access w v  = tt → access v u = tt → access w u = tt)
```

The Boolean type `bool` is used for truth values (i.e. either `tt` or `ff`).

Semantic Consequence. We have a forcing function which takes a model, a formula, and a world as inputs and returns a boolean value. Non-modal connectives are given truth-functionally and a formula $\Box p$ is true at a world w iff if $\mathcal{R}(w, v)$ then p is true at v, for all $v \in \mathcal{W}$:

```
def forces_form (M : model) : form σ → wrld σ → bool
| (#p)     := λ w, M.val p w
| (bot σ) := λ w, ff
```

```
| (p ⊃ q) := λ w, (bnot (forces_form p w)) ||(forces_form q w)
| (□p)    := λ w, if (∀ v ∈ M.wrlds, w ∈ M.wrlds →
M.access w v = tt → forces_form p v =tt) then tt else ff
```

This function can be extended to contexts in the obvious way:

```
def forces_ctx (M : model) (Γ :ctx σ) :wrld σ → bool :=
λ w, if (∀ p, p ∈ Γ → forces_form M p w =tt) then tt else ff
```

A formula p is a semantic consequence of a context Γ iff, for all $\mathcal{M}$ and $w \in \mathcal{W}$, Γ being true at w in $\mathcal{M}$ implies p being true at w in $\mathcal{M}$.

```
inductive sem_csq (Γ :ctx σ) (p :form σ) :Prop
| is_true (m : ∀ (M :model) (w : wrld σ),
forces_ctx M Γ w =tt → forces_form M p w =tt) : sem_csq

notation Γ '⊨s5' p :=sem_csq Γ p
```

3 The Completeness Theorem

In this section we formalize a proof of completeness with respect to the proof system and semantics developed in the previous sections.

Theorem 1 (Completeness). *For every context* Γ*, any formula* p *that follows semantically from* Γ *is also derivable from* Γ *in the modal logic S5. In symbols:*

$$\Gamma \vDash_{\mathrm{S5}} p \Longrightarrow \Gamma \vdash_{\mathrm{S5}} p$$

That is, every semantic consequence is also a syntactic consequence in S5.

The proof requires a non-constructive use of contraposition. We assume that both $\Gamma \vDash_{\mathrm{S5}} p$ and $\Gamma \nvdash_{\mathrm{S5}} p$ hold, and then derive a contradiction using the syntax to build a model $\mathcal{M} = \langle \mathcal{W}, \mathcal{R}, v \rangle$ (the canonical model) where Γ is true but p is false at a specific world in the domain.

We shall focus on sketching the formal argument for the following facts:

1. $\Gamma \cup \{\sim p\}$ has a maximal consistent extension $\Delta = \bigcup_{n \in \mathbb{N}} \Delta_n$, for

$$\Delta_0 := \Gamma \cup \{\sim p\}$$

$$\Delta_{n+1} := \begin{cases} \Delta_n \cup \{\varphi_{n+1}\} & \text{if } \Delta_n \cup \{\varphi_{n+1}\} \text{ is consistent} \\ \Delta_n \cup \{\sim \varphi_{n+1}\} & \text{otherwise} \end{cases}$$

2. There exists a canonical model where p is true at w iff $p \in w$;

Maximal Consistent Sets. We say that a context is maximal consistent if it is consistent and, moreover, for every formula expressible in the language, either it or its negation is contained in that context.

```
def is_max (Γ :ctx σ) :=is_consist Γ ∧ (∀ p, p ∈ Γ ∨ (∼p) ∈ Γ)
```

Our language is countable, so we can construct each Δ_{n+1} using natural numbers to run through the set of all formulas, deciding whether or not a number's corresponding formula (when it exists) is consistent with Δ_n or not. The enumerability of the language is expressed using `encodable` types, which are constructively countable types in Lean. Essentially, a type α is encodable when it has an injection `encode` $:\alpha \to$ `nat` and a (partial) inverse `decode` $:$`nat` $\to$ `option` α .

```
def insert_form (Γ :ctx σ) (p :form σ) :ctx σ :=
if is_consist (Γ ⸴ p) then ⸴ p else ⸴ ∼p

def insert_code (Γ :ctx σ) (n :nat) : ctx σ :=
match encodable.decode (form σ) n with
| none   := Γ
| some p := insert_form p
end

def maxn (Γ :ctx σ) :nat → ctx σ
| 0     := Γ
| (n+1) := insert_code (maxn n) n

def max (Γ :ctx σ) :ctx σ :=⋃ n, maxn Γ n
```

Before proceeding any further, we must show that Γ in contained in `max` Γ and that `max` Γ is maximal and consistent. For each `maxn` Γ `n` of the family of sets, we have Γ $\subseteq$ `maxn` Γ `n`. So Γ must also be contained in their union, `max` Γ. This proof argument produces a term of type:

```
lemma subset_max_self {Γ :ctx σ} :Γ ⊆ max Γ
```

Now, every formula must be in the enumeration somewhere, so suppose that the formula p has index `i`. By the definition of `maxn` Γ `i`, either p or $\sim p$ is a member of `maxn` Γ `i`, so one of them is a member of `max` Γ. Thus, we have a term

```
theorem mem_or_mem_max {Γ :ctx σ} (p :form σ) :p ∈ max Γ ∨ (∼p)
    ∈ max Γ
```

Assume for the sake of contradiction that Γ is consistent but `max` Γ is not. By structural induction on the proof tree, we prove that there exists an `i` such that `maxn` Γ `i` is inconsistent. However, each `maxn` Γ `i` preserves consistency. This gives a function

```
lemma is_consist_max {Γ :ctx σ} :is_consist Γ → is_consist (
    max Γ)
```

The above proof sketches are implemented purely by unfolding definitions and inductive reasoning. They consist of approximately 150 lines of code in `completeness.lean`. There is even a one-line short case-reasoning proof that maximal consistent sets are closed under derivability:

```
lemma mem_max_of_prf {Γ :ctx σ} {p :form σ} (h₁ :is_max Γ)
(h₂ :Γ ⊢S5 p) :p ∈ Γ :=
(h₁.2 p).resolve_right (λ hn, h₁.1 (prf.mp (prf.ax hn) h₂))
```

The Canonical Model Construction. We build the model as follows:

- $\mathcal{W}$ is the set of all maximal consistent sets of formulas;
- $\mathcal{R}(w, v)$ iff $\Box p \in w$ implies $p \in v$;
- $\mathsf{v}(w, p) = 1$ if $w \in \mathcal{W}$ and $p \in w$, for a propositional letter p.

We have to show that $\mathcal{R}$ is an equivalence relation. Reflexivity translates as follows: $\Box p \in w$ implies $p \in w$ for a given world $w \in \mathcal{W}$. But this is easy because w is closed under derivability (it is a maximal consistent set of formulas) and our proof system has modus ponens and axiom schema (t).

Proving symmetry requires more work. Given any worlds $w, v \in \mathcal{W}$, suppose first that $\Box\varphi \in w$ implies $\varphi \in v$ for all formulas φ, and suppose that $\Box p \in v$. We want to show that $p \in w$. Since $\Diamond\Box p \supset p$ is a theorem of S5 (see `syntax/lemmas.lean`) we just have to prove that $\Diamond\Box p \in w$, or, equivalently, that $\Box{\sim}\Box p \notin w$. By contraposition on our initial hypothesis, it suffices to show that ${\sim}\Box p \notin v$. But $\Box p \in v$ and v is consistent.

For transitivity, we must show that $p \in u$, on the assumptions that $\Box p \in w$, that $\Box\varphi \in w$ implies $\varphi \in v$, and that $\Box\varphi \in v$ implies $\varphi \in u$, for any formula φ. In other words, we want to show that $\Box\Box p \in w$. But this follows from modus ponens and axiom scheme (s4).

This model construction is represented by the Lean code

```
def domain (σ :nat) : set (wrld σ) :={w | ctx.is_max w}

def unbox (w : wrld σ) :wrld σ :={p | (□p) ∈ w}

def access : wrld σ → wrld σ → bool :=
λ w v, if (unbox w ⊆ v) then tt else ff

def val : fin σ → wrld σ → bool :=
λ p w, if w ∈ domain σ ∧ (#p) ∈ w then tt else ff

lemma mem_unbox_iff_mem_box {p : form σ} {w :wrld σ} :
p ∈ unbox w ↔ (□p) ∈ w :=⟨ id, id ⟩
```

What is here called `unbox` is a set operation which takes a set of formulas w as an input and returns the set of formulas p such that $\Box p$ is a member of w.

A useful lemma about this operation is that if p is deducible from `unbox` w then actually $\Box p \in w$. It can be proved by structural induction on the derivation using the necessitation rule, giving us a term:

```
lemma mem_box_of_unbox_prf {p : form σ} {w :wrld σ}
(H : w ∈ domain σ) :(unbox w ⊢s5 p) → (□p) ∈ w
```

Truth and Membership. To prove completeness, we first show that a formula is true at a world in the canonical model iff it is a member of that world:

```
lemma form_tt_iff_mem_wrld {p : form σ} :
∀ (w ∈ domain σ), (forces_form model w p) =tt ↔ p ∈ w
```

Here `model` is the canonical model defined in the previous section. To prove this, we use induction on the structure of the formula p.

In the proof mechanization, we use the `induction` tactic in the tactic mode. This tactic produces four goals, of which the fourth is the most relevant one. To prove it, we begin by assuming the inductive hypothesis for p. If w is a world, and, if it is a maximal consistent set of formulas, then, by unfolding the definition of truth of a formula at a world in a model, the biconditional statement becomes

```
⊢ (∀ (v :wrld σ),
v ∈ model.wrlds → w ∈ model.wrlds → model.access w v =tt →
    forces_form model w p =tt) ↔ (□p) ∈ w)
```

In the forwards direction, we assume that $\Box p$ is true at w in the canonical model and that $\sim\Box p \in w$. But then, by lemma `mem_box_of_unbox_prf`, the context `unbox` $w \cup \{\sim p\}$ is consistent and can be extended to a maximal consistent set (i.e. a world in the domain). It is accessible to w because `unbox` $w \subseteq$ `max` (`unbox` $w \cup \{\sim p\}$), so p should be true at w. But $p \notin$ `max` (`unbox` $w \cup \{\sim p\}$) because it is consistent.

For the backwards direction, assume that $\Box p \in w$. Given a maximal consistent set of formulas v and assuming that $\Box\varphi \in w$ then $\varphi \in v$ for all φ, we have to show that p is true at v in the model. By the inductive hypothesis, however, it suffices to show that $p \in v$, but this follows from $\Box p \in w$.

The Completeness Proof. We now complete our proof by putting together all the above pieces into 24 lines of code. Since we know by hypothesis that $\Gamma \nvdash_{\mathrm{S5}} p$, it follows that $\Gamma \cup \{\sim p\}$ is consistent–otherwise, if the false were deducible from it, we would have a contradiction by double negation elimination.

```
lemma consist_not_of_not_prf {Γ :form σ} {p :form σ} :
(Γ ⊬s5 p) → is_consist (Γ ∘ ∼p) :=λ hnp hc, hnp (mp dne (
    deduction hc))
```

Now assuming that $\Gamma \vDash_{\mathrm{S5}} p$, the basic idea for deriving the contradiction is that, as `max` $\Gamma \cup \{\sim p\}$ is a world in the canonical model, and each formula $\varphi \in \Gamma$ is true at that world, Γ is true as well. Clearly, p is not consistent with $\Gamma \cup \{\sim p\}$, so $p \notin$ `max` $\Gamma \cup \{\sim p\}$, meaning that p must be false at that world.

This allows us to prove the desired theorem

```
theorem completenss {Γ :form σ} {p :form σ} :(Γ ⊨s5 p) → Γ ⊢s5
```

Acknowledgments. Work supported in part by the AFOSR grant FA9550-18-1-0120. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the AFOSR.

References

1. Carneiro, M.: The lean 3 mathematical library (mathlib) (2018). https://robertylewis.com/files/icms/Carneiro_mathlib.pdf. International Congress on Mathematical Software
2. Coquand, C.: A formalised proof of the soundness and completeness of a simply typed lambda-calculus with explicit substitutions. Higher-Order Symb. Comput. **15**(1), 57–90 (2002)
3. Coquand, T., Huet, G.: The calculus of constructions. Inf. Comput. **76**(2–3), 95–120 (1988)
4. Doczkal, C., Smolka, G.: Constructive completeness for modal logic with transitive closure. In: Hawblitzel, C., Miller, D. (eds.) CPP 2012. LNCS, vol. 7679, pp. 224–239. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-35308-6_18
5. Doczkal, C., Smolka, G.: Completeness and decidability results for CTL in coq. In: Klein, G., Gamboa, R. (eds.) ITP 2014. LNCS, vol. 8558, pp. 226–241. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08970-6_15
6. From, A.H.: Epistemic logic. Archive of Formal Proofs, October 2018. https://devel.isa-afp.org/entries/Epistemic_Logic.html. Formal proof development
7. Herbelin, H., Lee, G.: Forcing-based cut-elimination for Gentzen-style intuitionistic sequent calculus. In: Ono, H., Kanazawa, M., de Queiroz, R. (eds.) WoLLIC 2009. LNCS (LNAI), vol. 5514, pp. 209–217. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-02261-6_17
8. Hughes, G.E., Cresswell, M.J.: A New Introduction to Modal Logic. Psychology Press (1996)
9. Negri, S.: Kripke completeness revisited. In: Acts of Knowledge: History, Philosophy and Logic: Essays Dedicated to Göran Sundholm, pp. 247–282 (2009)
10. Norell, U.: Dependently typed programming in Agda. In: Koopman, P., Plasmeijer, R., Swierstra, D. (eds.) AFP 2008. LNCS, vol. 5832, pp. 230–266. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-04652-0_5
11. Pfenning, F., Paulin-Mohring, C.: Inductively defined types in the calculus of constructions. In: Main, M., Melton, A., Mislove, M., Schmidt, D. (eds.) MFPS 1989. LNCS, vol. 442, pp. 209–228. Springer, New York (1990). https://doi.org/10.1007/BFb0040259
12. The Coq project: The coq proof assistant. http://www.coq.inria.fr (2017)
13. Tews, H.: Formalizing cut elimination of coalgebraic logics in Coq. In: Galmiche, D., Larchey-Wendling, D. (eds.) TABLEAUX 2013. LNCS (LNAI), vol. 8123, pp. 257–272. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-40537-2_22
14. Wu, M., Goré, R.: Verified decision procedures for modal logics. In: Harrison, J., O'Leary, J., Tolmach, A. (eds.) 10th International Conference on Interactive Theorem Proving (ITP 2019). Leibniz International Proceedings in Informatics (LIPIcs), vol. 141, pp. 31:1–31:19. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, Dagstuhl (2019). https://doi.org/10.4230/LIPIcs.ITP.2019.31. http://drops.dagstuhl.de/opus/volltexte/2019/11086