Extended version of:
Logic and Argumentation: Fourth International Conference,
CLAR 2021, Hangzhou, China, October 20–22
P. Baroni, C. Benzmüller, Y. N. Wáng (Eds.), pp. 459–467, 2021
doi:10.1007/978-3-030-89391-0_25

A Henkin-style completeness proof for the modal logic S5*

Bruno Bentzen

Carnegie Mellon University
Pittsburgh, Pennsylvania, USA
b.bentzen@hotmail.com

Abstract

This paper presents a recent formalization of a Henkin-style completeness proof for the propositional modal logic S5 using the Lean theorem prover [5]. The proof formalized is close to that of Hughes and Cresswell [11], 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 at https://github.com/bbentzen/mpl/ and has been typechecked with Lean 3.4.2.

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 [14] but the formal verification of completeness with respect to Kripke semantics is comparatively recent.

This paper presents a formalization of a Henkin-style completeness proof for the propositional modal logic S5 using the Lean theorem prover [5]. The proof is specific to S5, but, by forgetting the appropriate extra accessibility conditions (as described in [11]), the technique we use can be applied to weaker normal modal systems such as K, T, S4, and B. The formalization covers the syntax and semantics of S5, syntactic and semantic deduction theorem, structural rules (weakening, contraction, exchange), the recursive enumerability of the language, and soundness and completeness results. In total, it has approximately 1,500 lines of code, but only two thirds of it is required for the completeness proof. The full source code is available online at https://github.com/bbentzen/mpl/ and has been typechecked with Lean 3.4.2. At the time of writing, this is the latest stable release of Lean.

*The author wishes to thank Jeremy Avigad, Mario Carneiro, Rajeev Goré, and Minchao Wu for helpful suggestions and invaluable conversations on the topic covered herein. An early version of this development was presented at the Lean Together 2019, Amsterdam, January 7–11, 2019.

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 [3] uses ALF to give a constructive formal proof of soundness and completeness with respect to Kripke models for intuitionistic propositional logic with implication as the sole logical constant. Building on Coquand's work, a constructive completeness proof of Kripke semantics for intuitionistic logic with implication and universal quantification has been verified with Coq by Heberlin and Lee [10]. Also using Coq, Doczal and Smolka present a constructive formal proof of completeness with respect to Kripke semantics and decidability of the forcing relation for an extension of the basic modal logic K [6] and a variety of temporal logic [7]. In his formal verification of cut elimination for coalgebraic logics, Tews [19] provides a formalization of soundness and completeness proofs covering many different logics, including modal logic K.

However, to the best of our knowledge, the formalization of a Henkin-style completeness proof for propositional modal logic S5 proposed in this paper is the first of its kind.[1] Our proof is close to that of Hughes and Cresswell [11], 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 and based on a version of dependent type theory known as the Calculus of Inductive Constructions [16, 4]. Theorem proving in Lean can be done by constructing proof terms directly as in Agda [15], by using tactics (imperative commands that describe how to construct a proof term) as in Coq [18], or by mixing them together in the same proof environment. Like most proof assistants, Lean also supports classical reasoning, which is employed in the formalization along with the declaration of noncomputable constructions. Finally, the formalization also presupposes a few basic results on data structures which are not in the standard library, so we make use of `mathlib`, the library of formalized mathematics for Lean [2].

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

We start with the syntax and semantics of the modal logic we have formalized, namely, S5, the strongest of the five systems of modal logic proposed by Lewis and Langford [12]. In the next section, we give a proof sketch of completeness for the logic described here.

[1] In independent work done roughly at the same time the author first completed this formalization in 2018, Wu and Goré [20] 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 [8] formalized a Henkin-style completeness proof for system K in Isabelle.

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 choice of primitive operators gives rise to modal logics with a complex lattice structure [13], but it can also be used to do intuitionistic modal logic if one wishes to do so. This language can be very conveniently defined using inductive types in Lean, in which case the parsing tree of a formula is always made explicit—in fact, knowing that a certain formula is well-formed amounts to knowing whether it is well-typed term of that type.

```
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))
```

It is also possible to have notations for compound terms. Negation and possibility, for example, are encoded as definable logical and modal constants. One of the benefits of having the false as primitive rather than negation is that the statement that a certain set of premises (context) is consistent need not be dependent on particular choices of variables. In other words, we can write $\Gamma \nvdash_{S5} \bot$ instead of $\Gamma \nvdash_{S5}$ `p & ∼p` to express that Γ is consistent.

2.2 Contexts

Contexts are just sets of formulas, so, since we already have a type of formulas, implementing contexts does not require extra work. However, as there is a variety of set-like objects in Lean (lists, multisets, finsets, sets etc.), there are many possible design choices.

In an earlier version of the formalization, contexts were implemented using lists. A strong point of this approach is that one can reason by induction on contexts (lists are inductive types), allowing, among other things, a reduction of strong completeness to weak completeness via the deduction theorem [1]. Unfortunately, since lists must always be finite in size, there is no consistent way of reproducing maximal consistent extensions of contexts this way because every maximal consistent set is necessarily infinite.

Contexts are now defined using sets, which are functions of type `A → Prop`. Simply put, a set is just a predicate, a non-ordered collection that cannot contain duplicates (as a result, exchange and contraction are trivial structural rules of the proof system). For the purpose of stating more readable functions which look like logic textbook theorems, we have:

```
def ctx : Type := set (form σ)

notation `·` := {}
```

```
notation Γ `,` p :=set.insert p Γ
```

2.3 The proof system

We define a Hilbert-style axiomatic system, which is presented as a Mendelson system augmented with axiom schemes for K, T, S4, and B, and the necessitation rule as rule of inference. Such a proof system can be succinctly defined as follows in Lean:

```
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
```

Note that the use of the empty context $\cdot$ in the definition of `nec` (the necessitation rule) is essential to the completeness proof described in this paper. We shall return to it soon when discussing the deduction theorem and semantic consequence relation.

But before addressing those, it is worth pausing for a moment to give an illustration of how terms of the type of proofs defined above can be used to represent actual derivations in the modal logic S5. In particular, notice that the following term

```
lemma box_dne {p : form σ} :
· ⊢_S5 (□∼∼p) ⊃ (□p) :=
prf.mp (prf.k) (prf.nec (prf.dne))
```

translates as a proof tree of the following theorem

$$\dfrac{\dfrac{}{\Box(p \supset \sim\sim p) \supset (\Box p \supset \Box\sim\sim p)}\,(\mathrm{k}) \qquad \dfrac{\dfrac{}{p \supset \sim\sim p}\,(\mathrm{dne})}{\Box(p \supset \sim\sim p)}\,(\mathrm{nec})}{\Box p \supset \Box\sim\sim p}\,(\mathrm{mp})\ .$$

In some deductive systems for modal logic, the deduction theorem, which states that if a formula q is deducible from a set of assumptions $\Gamma \cup \{p\}$ then the implication $p \supset q$ is deducible from Γ, does not hold. In this formulation, it can be proven by a straightforward induction on the structure of the proof tree or, what amounts to the same thing, by an application of the elimination rule of the type of proofs defined above.

```
theorem deduction {Γ :ctx σ} {p q :form σ} :
(Γ , p ⊢_s5 q) → (Γ ⊢_s5 p ⊃ q) :=
begin
generalize eq : (Γ , p) =Γ',
intro h,
induction h; subst eq,
{ repeat {cases h_h},
```

```
exact id,
{ exact mp pl1  (ax h_h) } },
{ exact mp pl1 pl1 },
{ exact mp pl1 pl2 },
{ exact mp pl1 pl3 },
{ apply mp,
{ exact (mp pl2 (h_ih_hpq rfl)) },
{ exact h_ih_hp rfl } },
{ exact mp pl1 k },
{ exact mp pl1 t },
{ exact mp pl1 s4 },
{ exact mp pl1 b },
{ exact mp pl1 (nec h_h) }
end
```

The proof of all cases follows from a simple application of modus ponens to axiom scheme (1), except for the modus ponens case that requires axiom scheme (2). The presence of the necessitation rule in the form of `nec` does not impose any difficulty in the proof because the context restriction in the antecedent gives us a stronger inductive hypothesis.

As noted in [9], whether the deduction theorem fails or not in modal logics which have the necessitation rule is a consequence of design choice. The necessitation rule, which informally says that if p is a theorem of the system then so is $\Box p$, can be implemented in many ways. As can be seen above, we restrict its applicability to theorems by only allowing formulas that are derivable under no assumptions in the antecedent of `nec`. Another possibility is having a more general (i.e. unrestricted) formulation of the necessitation rule, which extends the range of application of `nec` to formulas that are derivable under assumptions. That would mean having a constructor such as `nec'` instead of `nec` in the formalization:

```
| nec' {Γ} {p} (h :prf Γ p) :prf Γ (□p)
```

while this rule can be useful and, indeed, the resulting system can be shown to be complete with respect to a global version of semantic consequence [17] (see section 2.4.2), the deduction theorem does not hold in the presence of it.

2.4 Semantics

2.4.1 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.

Typically, there are no constraints on what kind of objects the members of $\mathcal{W}$ (possible worlds) should be. For our purposes, however, it is useful to let them be sets of formulas instead of arbitrary objects (there is no loss of generality in the resulting semantics). We thus have:

```
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. In the following 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 in the formalization of truth values (i.e. either `tt` or `ff`).

2.4.2 Semantic consequence

We have a recursively defined forcing function which takes a model, a formula, and a world as inputs and returns a boolean value. It can be defined by induction on the structure of the formula involved as follows:

```
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
```

The definition is a routine adaptation of that found in traditional modal logic textbooks [11]. Non-modal connectives are given truth-functionally and the necessity operator is defined by stating that a formula $\Box p$ is true at a world w iff if $\mathcal{R}(w, v)$ then p is true at v, for all $v \in \mathcal{W}$.

This function can be extended to contexts in the obvious way: we say that a context is true at a world in a model if each formula of that context is true at that world and in that model.

```
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 *local* semantic consequence of a context Γ iff, for all models $\mathcal{M}$ and for all worlds $w \in \mathcal{W}$, the fact that Γ is true at w in $\mathcal{M}$ implies that p is 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
```

Say that a formula (context) is true in a model $\mathcal{M}$ if that formula (context) is true at all worlds in that model. A formula p is a *global* semantic consequence of a context Γ iff, for any model $\mathcal{M}$, if Γ is true in $\mathcal{M}$ then p is true in $\mathcal{M}$. It is easy to see that local semantic consequence implies global semantic consequence, so the global version of completeness implies the local version. The converse does not hold. In particular, note that $\Box p$ is a global—but not a local—semantic consequence of $\{p\}$. The formalization includes local but not global semantic consequence. It is not hard to encode, but the global version of completeness is actually false for our proof system.[2]

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. First, we state the completeness theorem. Second,

[2] Unless `nec` is replaced with `nec'`. See [17] for an informal proof.

a general outline of the proof is presented. Next, we give an informal explanation of each individual proof step followed by a description of its respective implementation.

Theorem 3.0.1 (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 here requires full contraposition and it is thus non-constructive. More specifically, the idea is to 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. To complete the proof we shall need some additional lemmas, many of which can be easily proven by induction.

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

1. $\Gamma \cup \{\sim p\}$ has a maximal consistent extension Δ defined as follows:

$$\begin{aligned} \Delta_0 &:= \Gamma \cup \{\sim p\} \\ \Delta_{n+1} &:= \begin{cases} \Delta_n \cup \{\varphi_{n+1}\} & \text{if } \Delta_n \cup \{\varphi_{n+1}\} \text{ is consistent} \\ \Delta_n \cup \{\sim \varphi_{n+1}\} & \text{otherwise} \end{cases} \\ \Delta &:= \bigcup_{n \in \mathbb{N}} \Delta_n \end{aligned}$$

   (i.e. Δ is consistent, maximal and $\Gamma \cup \{\sim p\} \subseteq \Delta$)

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

3.0.1 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)
```

As our language is countable, it is possible to 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 (another alternative is to generalize this construction to languages of arbitrary cardinalities using Zorn's lemma instead).

The enumerability of the language is expressed using `encodable` types, which are constructively countable types. Essentially, a type α is encodable when it has an injection `encode :α → nat` and a (partial) inverse `decode :nat → option α`.

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

def insert_code (Γ :ctx σ) (n :nat) : ctx σ :=
match encodable.decode (form σ) n with
| none   := Γ
| some p := insert_form Γ 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 `max` Γ is the maximal consistent extension of Γ, that is, that Γ in contained in `max` Γ and that `max` Γ is maximal and consistent.

First, we note that, for each `maxn` Γ `n` of the family of sets, we have Γ ⊆ `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
```

Second, we observe that 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
```

Third, 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))
```

3.0.2 The canonical model construction

In this section we build the canonical model for S5 from the deductive system itself. For that we specify a domain, an accessibility relation, a valuation function, and proofs of reflexivity, symmetry and transitivity for the accessibility relation. 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.

For that to be a well-defined model, we must 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 rather trivial but still quite 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. The proof argument is straightforward but, because we shall need this fact in the next section, it is convenient to have the formal proof presented here:

```
lemma mem_box_of_unbox_prf {p : form σ} {w :wrld σ}
(H : w ∈ domain σ) :(unbox w ⊢s5 p) → (□p) ∈ w :=
begin
generalize eq : unbox w = Γ',
intro h, induction h; subst eq,
{ assumption },
repeat { apply ctx.mem_max_of_prf H,
apply prf.nec,
apply prf.pl1 <|> apply prf.pl2 <|> apply prf.pl3 },
{ apply ctx.mem_max_of_prf H,
refine prf.mp (prf.ax _) (prf.ax (h_ih_hp rfl)),
exact (ctx.mem_max_of_prf H)
(prf.mp prf.k (prf.ax (h_ih_hpq rfl))) },
{ apply ctx.mem_max_of_prf H,
exact prf.nec prf.k },
{ apply ctx.mem_max_of_prf H,
exact prf.nec prf.t },
{ apply ctx.mem_max_of_prf H,
exact prf.nec prf.s4 },
{ apply ctx.mem_max_of_prf H,
exact prf.nec prf.b },
{ apply ctx.mem_max_of_prf H,
apply prf.nec (prf.nec h_h) }
end
```

3.0.3 Truth and membership

In order to prove completeness, we first show that truth is membership in the canonical model, that is, 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
```

where `model` is the canonical model defined in the previous section. To prove this, we use induction on the structure of the formula p. So, we will show that the result holds when p is a propositional letter $\#p$, the false $\bot$, an implication $p \supset q$ or a necessary proposition $\Box p$.

In the proof mechanization, we use the `induction` tactic in the tactic mode. This tactic produces four goals (as displayed below).

```
case form.atom
σ :ℕ,
p : fin σ
⊢ ∀ (w :wrld σ),
w ∈ domain σ → forces_form model w #p =tt ↔ #p ∈ w)
```

The first goal is a base case. If p is a propositional letter the biconditional statement is clearly true, because, by definition of the valuation function, this is what it means for a propositional letter to be true at a world in the canonical model.

```
case form.bot
σ :ℕ,
⊢ ∀ (w :wrld σ),
w ∈ domain σ → forces_form model w ⊥ =tt ↔ ⊥ ∈ w)
```

Proving the second goal is easy. Both conditional statements are vacuously true in both directions for the false, because w is consistent and, by definition, the false cannot be true at a world in a model.

```
case form.impl
σ :ℕ,
p q : form σ,
hp :
∀ (w :wrld σ),
w ∈ domain σ → forces_form model w p =tt ↔ p ∈ w),
hq :
∀ (w :wrld σ),
w ∈ domain σ → forces_form model w q =tt ↔ q ∈ w)
⊢ ∀ (w :wrld σ),
w ∈ domain σ → forces_form model w (p ⊃ q) =tt ↔ (p ⊃ q) ∈ w)
```

For the third goal, we assume the inductive hypothesis that the result holds for p and q, and then show that either p is false or q is true at w in the canonical model iff $p \supset q \in w$. The proof follows from case reasoning and the closure under derivability of possible worlds.

```
case form.box
σ :ℕ,
p : form σ,
hp :
∀ (w :wrld σ),
w ∈ domain σ → forces_form model w p =tt ↔ p ∈ w)
⊢ ∀ (w :wrld σ),
w ∈ domain σ → forces_form model w (□p) =tt ↔ (□p) ∈ w)
```

To prove the fourth goal, 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 $\Box p \in w$.

3.0.4 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 following theorem

```
theorem completenss {Γ :form σ} {p :form σ} :
(Γ ⊨_s5 p) → Γ ⊢_s5
```

4 Conclusion

We presented a computer formalization of the completeness of the modal logic S5 using a Henkin-style argument, as given in traditional textbooks such as [11]. An interesting direction for further work would be to formally verify the global version of completeness for the alternative presentation of S5 obtained by replacing `nec` with `nec'`. Another natural step for continuing the work presented here is to present completeness proofs for different modal logics.

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] Leran Cai, Ambrus Kaposi, and Thorsten Altenkirch. Formalising the Completeness Theorem of Classical Propositional Logic in Agda (Proof Pearl). URL: https://akaposi.github.io/proplogic.pdf, 2015.

[2] Mario Carneiro. The Lean 3 Mathematical Library (mathlib). URL: https://robertylewis.com/files/icms/Carneiro_mathlib.pdf, 2018. International Congress on Mathematical Software.

[3] Catarina Coquand. A formalised proof of the soundness and completeness of a simply typed lambda-calculus with explicit substitutions. *Higher-Order and Symbolic Computation*, 15(1):57–90, 2002. URL: https://doi.org/10.1023/A:1019964114625.

[4] Thierry Coquand and Gérard Huet. The calculus of constructions. *Information and Compututation*, 76(2-3):95–120, 1988. URL: https://hal.inria.fr/inria-00076024/document.

[5] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris Van Doorn, and Jakob von Raumer. The lean theorem prover (system description). In A. Felty and A. Middeldorp, editors, *International Conference on Automated Deduction*, pages 378–388, Cham, 2015. Springer. URL: https://doi.org/10.1007/978-3-319-21401-6_26.

[6] Christian Doczkal and Gert Smolka. Constructive completeness for modal logic with transitive closure. In C. Hawblitzel and D. Miller, editors, *International Conference on Certified Programs and Proofs*, pages 224–239, Berlin, Heidelberg, 2012. Springer. URL: https://doi.org/10.1007/978-3-642-35308-6_18.

[7] Christian Doczkal and Gert Smolka. Completeness and decidability results for CTL in Coq. In G. Klein and R. Gamboa, editors, *International Conference on Interactive Theorem Proving*, pages 226–241, Cham, 2014. Springer. URL: https://doi.org/10.1007/978-3-319-08970-6_15.

[8] Asta Halkjær From. Epistemic logic. *Archive of Formal Proofs*, October 2018. https://devel.isa-afp.org/entries/Epistemic_Logic.html, Formal proof development.

[9] Raul Hakli and Sara Negri. Does the deduction theorem fail for modal logic? *Synthese*, 187(3):849–867, 2012. URL: https://doi.org/10.1007/s11229-011-9905-9.

[10] Hugo Herbelin and Gyesik Lee. Forcing-based cut-elimination for Gentzen-style intuitionistic sequent calculus. In Ono H., Kanazawa M., and de Queiroz R., editors, *International Workshop on Logic, Language, Information, and Computation*, pages 209–217, Berlin, Heidelberg, 2009. Springer. URL: https://doi.org/10.1007/978-3-642-02261-6_17.

[11] George Edward Hughes and Max J Cresswell. *A new introduction to modal logic*. Psychology Press, 1996.

[12] Clarence Irving Lewis and Cooper Harold Langford. Symbolic Logic. The Century Company, New York, 1932, 1959.

[13] David Makinson. A warning about the choice of primitive operators in modal logic. *Journal of philosophical logic*, 2(2):193–196, 1973. URL: https://www.jstor.org/stable/30226058.

[14] Sara Negri. Kripke completeness revisited. *Acts of Knowledge: History, Philosophy and Logic: Essays Dedicated to Göran Sundholm*, pages 247–282, 2009. URL: https://www.mv.helsinki.fi/home/negri/gkcrev.pdf.

[15] Ulf Norell. Dependently typed programming in Agda. In P. Koopman, R. Plasmeijer, and D. Swierstra, editors, *International School on Advanced Functional Programming*, pages 230–266, Berlin, Heidelberg, 2008. Springer. URL: https://doi.org/10.1007/978-3-642-04652-0_5.

[16] Frank Pfenning and Christine Paulin-Mohring. Inductively defined types in the Calculus of Constructions. In *International Conference on Mathematical Foundations of Programming Semantics*, pages 209–228. Springer, 1989.

[17] Sally Popkorn. *First steps in modal logic*. Cambridge University Press, 1994.

[18] The Coq project. The coq proof assistant. URL: http://www.coq.inria.fr, 2017.

[19] Hendrik Tews. Formalizing cut elimination of coalgebraic logics in Coq. In D. Galmiche and D. Larchey-Wendling, editors, *International Conference on Automated Reasoning with Analytic Tableaux and Related Methods*, pages 257–272, Berlin, 2013. Springer. URL: https://doi.org/10.1007/978-3-642-40537-2_22.

[20] Minchao Wu and Rajeev Goré. Verified Decision Procedures for Modal Logics. In J. Harrison, J. O'Leary, and A. Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, pages 31:1–31:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik. URL: https://doi.org/10.4230/LIPIcs.ITP.2019.31.