Formalization of modal logic S5 in the Coq proof assistant

Bachelor thesis

University of Groningen

August 9, 2022

Author:
Lubor Budaj

First supervisor:
Revantha Ramanayake

Second supervisor:
Dan Frumin

Abstract

In this thesis, we present our formalization of modal logic S5 in Coq. Firstly, we define a formula and a model as a Kripke frame. S5 is a modal logic that has an equivalence assumption on the accessibility relation in the model. Then we define an interpretation of a model with respect to Kripke semantics and a proof system. As the basis of our proof system, we use a Hilbert system. Secondly, we show the proofs of soundness and completeness. The soundness property shows that proving theorems in our proof system makes sense - we can infer only valid theorems. Completeness means that everything that is true is provable. Together, these two properties show that our proof system is suitable for proving theorems of S5. We proved the completeness of S5 using a canonical model proof. In order to do that we first had to define concepts of a consistent set of formulas and a maximal consistent set. Then we showed that formula is a countable type, allowing us to define a maximal consistent completion of a set and the canonical model. Other modal logics that have weaker assumptions about the properties of their accessibility relation, such as K, T, or S4, can be formalized in a similar way.

Contents

1 Introduction

Mathematical formalization [10] is a description of a formal system and its interpretation in a formal language. As part of the formalization, we can prove various properties of the system. Formalization can be done with proof assistant software, allowing for verification of the proofs using a computer. In this thesis, we use Coq [14] proof assistant to formalize modal logic S5. Our goal is to find the most effective way to achieve this in Coq. Firstly, we define structures to represent a formula and a model for S5. Then, we define two concepts, validity and proof of a formula. We define validity (or truth) with respect to Kripke semantics - we use a model to evaluate the truth value of a formula. Validity and proof are related by the two theorems - soundness and completeness.

Soundness of a proof system states that everything that is provable is true. This is an essential property of any proof system. Suppose this was not the case. We would be able to prove false statements, effectively making the proof system useless. The second property we will prove is completeness. Completeness states that everything that is valid is provable. Although this property is not as essential as soundness - some formal systems are useful while being incomplete - if the system is complete, it provides valuable information for anyone using the proof system. Completeness ensures that if the user thinks their proposition is true, then there exists a proof, and finding the proof is not a useless task. The following example illustrates the significance of completeness. Suppose we have a formal system in which we can prove an instance of a single axiom but we do not define any other axioms or inference rules. This system can be sound but is not complete nor useful.

The thesis is structured in the following way. First, we provide formal definitions both as mathematical statements and in Coq. Then, from these definitions, we propose theorems and prove them. We do not show complete proofs for all the theorems. The reason for that is twofold. Firstly, our goal is not to develop a new way of proving the soundness and the completeness of S5, but to use already existing proofs and modify them, such that they are suitable for use in Coq. Secondly, the proof of completeness is rather lengthy. We need to prove many sub-lemmas, many of which are simply not that interesting. For example, often in the proofs by induction, there are many unattractive cases. On the contrary, in our commentary, we focus on the interesting parts of the proofs. Most of the proofs in this thesis are informal. The reason for that is that this thesis is supposed to be read in a complementary way to the Coq code, which includes the formal proofs. All the proofs we show in this thesis follow the same idea as our proofs in Coq. The source code is available online at https://github.com/fondefjobn/S5-Formalization-in-Coq/.

We divide this thesis into six sections. First, as part of the introduction, we summarize the existing research, introduce Coq and provide a motivation for this project. Afterward, in section 2 we explain Coq prerequisites for this project. Experienced Coq users can feel free to skip or skim through this section. The following three sections are the main part of the thesis. In section 3 we describe the design of structures we use to represent a formula, a model, validity, and proof in S5. In section 4 the proof of the soundness of our proposed proof system is presented. In section 5 we first introduce the necessary concepts to make a proof of the completeness and proceed with the proof itself. Lastly, in section 6 we conclude our work and touch on future work.

1.1 Coq

Coq is an interactive theorem prover, which can be used for formalizing mathematical concepts or proving correct executions of programs. Coq is based on the Calculus of Constructions, which is a constructive theory of mathematical propositions by Coquand and Huet [5] from 1984. Subsequently, Coq has been developed at INRIA (French Institute for Research in Computer Science and Automation) and is still in active development.

There are two types of software tools for making logical proofs - automated theorem provers and proof assistants. Automated theorem provers can automatically determine the validity of a logical proposition, however, this is often limited to simple propositions [13]. Proof assistants, such as Coq, do not automatically determine the validity of the proposition. Instead, they are tools that help automate the routine tasks of a proof. The user needs to define a proof themselves instead of the program finding the proof for the user. This is achieved by the use of various proof tactics. In Coq, the tactics can be applied interactively, hence the user does not need to define the whole proof immediately. The tactics can be applied one by one and the user can see the current proof state at any given stage of the proof. In general, when making proofs in Coq, we use backward reasoning [11] - we apply tactics to update the goal. However, it is also possible to work in a forward way by applying the tactics directly in the hypotheses or by introducing new hypotheses.

As Coq is a proof assistant, it is desirable to ensure the correctness of the proofs. Hence, to avoid any side effects of functions that come with different program states, Coq uses a functional paradigm. Mathematical formalization in Coq can be divided into three parts [11]. Firstly, we define the structures to represent modal logic. In this part, we use standard functional programming language features such as inductive data types, pattern matching, or higher-order functions. Secondly, we make mathematical statements using these definitions with the help of built-in logical connectives. Lastly, there is a proof part, where we prove the propositions we previously defined. This can be achieved by applying various proof tactics. It is also possible to define new tactics ourselves.

Coq has various features which help prove theorems. We can use lemmas to define sub-proofs of larger proofs or to define a proof for a common pattern. We can make use of various proof techniques - proof by rewriting, proof by case analysis, proof by contradiction, proof by induction, or others. Often, when we apply a tactic instead of a single goal we are left with multiple goals, such as with proofs by induction. In order to keep the proofs clear, Coq supports bullet symbols, using which we can structure the different cases of the proof. Similarly, we can structure the proof with curly braces with the same effect. Coq comes with many automated tactics such as *auto* or *tauto*, however, we decided not to use these tactics in order to keep the proofs as clear as possible. Similarly, even though Coq comes with a powerful type inference mechanism, most of the time, we define the types explicitly in order to keep the code more readable.

1.2 Motivation

Making proofs in modal logic is useful because it allows verifying sentences that make use of modal verbs 'may' or 'must', which are common in a human language but are not in the scope of standard propositional logic. Furthermore, using modal logic is a good way to reason about many relational structures [3]. Hence, it is useful in many fields such as philosophy or artificial intelligence.

The standard way to make a formal proof is by hand - in other words with pen and paper. However, there are many advantages of using a proof assistant, such as Coq, instead. Firstly, when making a proof by hand it is very easy to make a mistake. This mistake can end up being spotted much later, forcing us to redo a large part of the proof, or even not found at all. On the other hand, it is not possible to do such a misstep in Coq. Coq allows only valid uses of proof tactics. By default, any given proposition has to be proven before we can continue. It is possible to manually admit an unproven proposition, which can be useful when we want to finish a certain part of the proof later. Contrary to making a mistake, this is always done with the knowledge of the user. Secondly, Coq allows us to make proofs step-by-step. At any given state of the proof, we can see all the hypotheses and the goals. We can apply the tactics one by one to see the changes each tactic made to the goals and hypotheses. Lastly, we can easily make changes to the structures we use and we will immediately see which proofs are no longer valid and need to be updated.

1.3 Related Work

There are several works in our area of interest.

Bednarska and Indrzejczak [1] discuss different implementations of hypersequent calculi for S5 and their advantages and disadvantages. However, they provide only proof sketches and do not include a proof assistant formalization. Fitting [7] made a tableau system for S5 that includes a completeness proof. However, the proof is only for weak completeness. Furthermore, this work does not include a formalization in a proof assistant.

Doorn [15] made a formalization of propositional logic using a Hilbert system in Coq. It includes both soundness and completeness proofs. Nevertheless, it does not include any proof of properties of modal logic. Philipse [12] discusses the formalization of system S5^n, which is a multi-agent system with a separate relation for each agent. However, their focus is on distributed knowledge proofs. Their work does not include soundness and completeness proofs. Doczkal and Smolka [6] formalized K*, extensions of K with star modalities, in Coq. However, compared to our work, they focus on computational decidability rather than completeness. Hence, they made several design decisions differently compared to us. Secondly, their proof is for a different type of modal logic. Wu and Goré [16] provide formalization for modal logics K, T, and S4 in Lean proof assistant. Their formalization includes both soundness and completeness proofs. Nevertheless, their formalization uses the tableau method and does not include proof for S5. Bentzen [2]. formalized S5 in Henkin-style proof in a proof assistant Lean. Their proof includes both soundness and completeness. This work is the closest related work to ours.

2 Coq Preliminaries

2.1 Basic Types

There are three basic types in Coq - *Prop*, *Type* and *Set*. *Prop* is used to define logical propositions. In a proof, the hypotheses and the goals are of this type. The instances of propositions can be used with all standard logical connectives to define more complex logical statements. In general, if we want to prove a statement or use it with a logical connective, it needs to be of type proposition. Since propositions are used with logical connectives, it might be intuitive to think of propositions as booleans in other programming languages, however, this is not the case. The difference is that there are only two possible instances of a boolean - true and false. On the other hand, other than True and False there are infinitely many possible instances of a proposition. Furthermore, compared to instances of boolean, it is not possible to simplify a proposition. Instead, to deal with logical connectives we have to use various introduction and elimination tactics to make a progress in a proof, or one of the previously defined theorems. *Type* is used as a basic type to define mathematical structures. This is true for *Set* too, but we did not use it in the development.

2.2 Sets

Our next goal is to define a mathematical notion of a set (not to be confused with *Set* type) in Coq. Sets in Coq are known as Ensembles and are defined in the library of the same name, nevertheless, we define sets ourselves as it gives us more flexibility on how we use them.

```
Definition set T := T → Prop.
```

Even though in our formalization we use only a single type of set - a set of formulas - in the future we might want to use this definition for other types. Hence, we do not want to define a set for a particular type. We solve this using polymorphism. In this way, we can use a set with any type. We define a set as a function of type to a proposition. This approach is very flexible. We can define a set as any correctly-typed function and in this way we are even able to define infinite sets. For example, in the following way, we can define a set of all numbers larger than three.

```
Definition set_larger_than_3 :=
  fun (x : nat) ⇒ x > 3.
```

We define the set as a function with an argument x of type natural number. This function returns a proposition that x is larger than 3. If we can prove this proposition for x, we can prove that x belongs to the set. Next, we want to add definitions for different types of sets and define a subset.

```
Variable T : Type.

Definition empty_set : set T :=
  fun (x : T) ⇒ False.

Definition subset (F G : set T) : Prop :=
  forall x, F x → G x.

Definition add_singleton (G : set T) (a : T) : set T :=
  fun x ⇒ x = a ∨ G x.
```

We define an empty set as a function that returns the proposition False for any argument. In other words, if we want to prove that x is an element of the empty set we will need to prove False. We define a subset as a proposition with two arguments, such that all elements of the first set are the elements of the second set. Lastly, we define a union of a set with a singleton set. Here we use disjunction - the element is either a singleton or is in the set. Similarly, we could define other set operations such as intersection or union but they are not needed for our project.

3 Modal Logic S5

Modal logic is an extension of propositional logic. It introduces the notions of possible worlds, necessity, and possibility. A world v is accessible from a world u if there is a relation from u to v. A proposition is necessary [4] if it holds in all accessible worlds, possible if it holds in at least one world.

3.1 Syntax

Modal logic contains all standard connectives of propositional logic - $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction), $\rightarrow$ (implication) and $\leftrightarrow$ (biconditional). In addition, there are two new unary operators $\Box$ (box) and $\Diamond$ (diamond) [9], to express necessity and possibility respectively. In the following sections, we will use $\Rightarrow$ and $\iff$ as meta-level implication and biconditional. These are not the operators of S5.

Definition 3.1.

```
Definition var := nat.

Inductive form : Type :=
  | F_
  | Var (v : var)
  | Impl (f1 : form) (f2 : form)
  | Box (f : form).

Definition Neg (f : form) := Impl f F_.
Definition T_ := Neg F_.
Definition Disj (f1 f2 : form) := Impl (Neg f1) f2.
Definition Conj (f1 f2 : form) := Neg (Disj (Neg f1) (Neg f2)).
Definition BiImpl (f1 f2 : form) := Conj (Impl f1 f2) (Impl f2 f1).
Definition Diamond (f : form) := Neg (Box (Neg f)).
```

In Coq, a formula can be neatly defined as an inductive data type. Firstly, we realize that we do not need a constructor for each connective. Instead, we can use the fact that we can express the connectives in terms of each other. The advantage of this approach is that if we do a pattern matching or a proof by induction on this data type, there are going to be fewer cases to consider. To define all the connectives of propositional logic we only need to define false and implication constructors. The implication constructor takes two arguments of a formula type false takes no arguments. For modal logic operands, we define only the constructor box, which takes one formula as an argument. Diamond is defined in terms of box and negation. Lastly, we need a constructor for a variable. For simplicity, we decided to use natural numbers type as an argument for variable, but in principle, we could have used any infinite type.

3.2 Model

Other than being able to construct well-formed formulas, we want to be able to determine their validity, given a model. As a model, in S5 we use a Kripke frame.

Definition 3.2. *A Kripke frame is a tuple $\mathfrak{M} = (W, R, V)$ where W is the set of all possible worlds, R is the accessibility relation on W, and V is the valuation function for a world and a variable.*

```
Record model : Type := Model {
  world :> Type;
  rel : world → world → Prop;
  val : world → var → Prop;
  eq : equiv world rel
}.
```

In order to define a tuple in Coq, we use a record type. The first element of the tuple is a world. We define a world as a type - we use polymorphism again. If we were to use the model structure, we can choose what type the world is going to be. As we use world in definitions of other elements of the tuple, they are dependent on this choice. Secondly, an accessibility relation is defined as a function to proposition with two arguments of the type world we previously defined. The idea behind this definition is the same as the notion behind definitions of sets in section 2, the only difference being that we now have an additional argument that is used to define the function. Next, there is a valuation function. It is defined as a function to proposition with arguments being a world and a variable. The notion is very similar to the previous member, but here we have arguments of two different types. In addition to these three members, in Coq, we also define a proposition stating the proof that the relation is equivalent. This is a necessary condition for any model to be a S5 frame, as the model for S5 is defined only for equivalent accessibility relations. This is the only part we define explicitly. We define it using the function equiv from the library Relation_Definitions. Contrary to the mathematical definition of the Kripke frame, we do not need to define a set of worlds as part of the tuple. That is because all the worlds used are defined anyway in the relation and valuation functions.

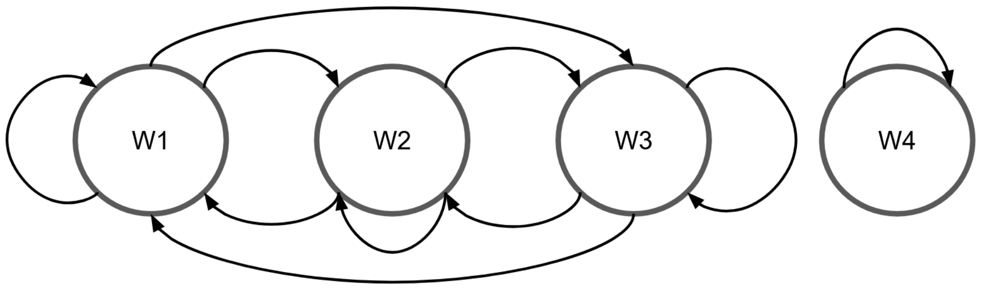

Figure 1: Equivalence relation

We distinguish many types of modal logics depending on the property of the relation between the worlds. The property of S5 relation is equivalence - that is reflexivity, transitivity and symmetry. We define these properties formally.

Definition 3.3. *A relation R is reflexive if only if* $\forall x \in R : (x, x) \in R$

Definition 3.4. *A relation R is transitive if only if* $\forall x, y, z \in R : ((x, y) \in R$ *and* $(y, z)) \in R \Rightarrow (x, z) \in R$

Definition 3.5. *A relation R is symmetric if only if* $\forall x, y \in R : (x, y) \in R \Rightarrow (y, x) \in R$

Definition 3.6. *A relation is equivalent if only if it is reflexive, transitive and symmetric.*

An example of an equivalence relation can be seen in Figure 1. In the figure, the labeled circles represent different worlds. A relation from a world u to a world v is represented as an arrow from u to v. Note that there are no other worlds in this relation.

S5 is a modal system with one of the strongest reasoning principles with respect to accessibility relation. Hence, if a certain proof does not make use of all properties of S5, the proof can still be valid in other systems [2] such as K, T, or S4. These systems use some, but not all of the relational properties of S5.

Having defined the model, we define truth in a world in the model. Here, we use α and β to denote the formulas and w and v to denote the worlds.

Definition 3.7.

$\mathfrak{M}, w \models \alpha$ iff $V(w, \alpha)$
$\mathfrak{M}, w \models \top$
$\mathfrak{M}, w \not\models \bot$
$\mathfrak{M}, w \models \neg\alpha$ iff $\mathfrak{M}, w \not\models \alpha$
$\mathfrak{M}, w \models \alpha \wedge \beta$ iff both $\mathfrak{M}, w \models \alpha$ and $\mathfrak{M}, w \models \beta$
$\mathfrak{M}, w \models \alpha \vee \beta$ iff either $\mathfrak{M}, w \models \alpha$ or $\mathfrak{M}, w \models \beta$
$\mathfrak{M}, w \models \alpha \rightarrow \beta$ iff if $\mathfrak{M}, w \models \alpha$ then $\mathfrak{M}, w \models \beta$
$\mathfrak{M}, w \models \alpha \leftrightarrow \beta$ iff $\mathfrak{M}, w \models \alpha$ if only if $\mathfrak{M}, w \models \beta$
$\mathfrak{M}, w \models \Box\alpha$ iff for every $v \in W$, if $(w, v) \in R$ then $\mathfrak{M}, v \models \beta$
$\mathfrak{M}, w \models \Diamond\alpha$ iff exists $v \in W$, $(w, v) \in R$ and $\mathfrak{M}, v \models \beta$

```
Fixpoint interpret (f : form) (m : model) (w : m) : Prop :=
  match f with
  | F_ ⇒ False
  | Var x ⇒ val w x
  | Impl f1 f2 ⇒ interpret f1 m w → interpret f2 m w
  | Box f ⇒ forall (v : m), rel w v → interpret f m v
  end.
```

Firstly, the Coq definition uses the keyword Fixpoint. This indicates that the function is recursive. Secondly, there are three arguments. A formula, a model, and a world from this model. We ensure that the world is from this model by defining the instance of the model as its type. The type of the function is a proposition, hence we will be able to reason about it using Coq connectives in later proofs. In the body of the function, we use pattern matching for a formula in a similar way as in the mathematical definition. However, as we defined it using only four constructors for a formula, there are only four cases to consider. For each case, the proposition returned follows directly from the mathematical definition.

3.3 Proof System

Definition 3.8.

1. $\alpha \to (\beta \to \alpha)$
2. $(\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma))$
3. $(\neg\alpha \to \neg\beta) \to (\beta \to \alpha)$

K. $\Box(\alpha \to \beta) \to (\Box\alpha \to \Box\beta)$
T. $\Box\alpha \to \alpha$
4. $\Box\alpha \to \Box\Box\alpha$
B. $\alpha \to \Box\Diamond\alpha$

Definition 3.9. *Modus ponens is an inference rule such that if there is $\alpha \to \beta$ and α, we can infer β.*

Definition 3.10. *Necessitation is an inference rule such that if there is α, we can infer $\Box\alpha$*

We define a Hilbert-style proof system with seven axioms and two inference rules. As the basis, we use a minimal Hilbert system for propositional logic defined with three axioms and one inference rule modus ponens, which is derived from Lukasiewicz's systems [8]. We chose this system because it is the most concise standard system we found. For any basic modal logic, we first need a distributivity axiom K and the necessitation inference rule. Then, in order to define the remaining axioms of S5, we have 2 choices. Either we can define it using axioms T, 4, and B, or T and 5. These two axiom schemes are equivalent. We chose the former scheme as it gives us more flexibility if we were to define another type of modal logic. We can think of each modal axiom (except K) as a property of the relation in the model - T is for the reflexive relations, 4 is for the transitive relations and B is for the symmetric relations. If we have a relation with all of these properties, it is an equivalence relation, which is needed for S5. The other axioms scheme defines S5 too. In that case, T is the reflexive property of the relation and 5 is the euclidean, which together form an equivalence relation too.

```
Definition form_set := set form.

Inductive ax_s5 (G : form_set) : form → Prop :=
  | a_0 (f : form) : G f → ax_s5 G f
  | a_1 (f g : form) : ax_s5 G (Impl f (Impl g f))
  | a_2 (f g h : form) : ax_s5 G (Impl (Impl f (Impl g h)) (Impl (Impl f g) (Impl f h)))
  | a_3 (f g : form) : ax_s5 G (Impl (Impl (Neg f) (Neg g)) (Impl g f))
  | a_k (f g : form) : ax_s5 G (Impl (Box (Impl f g)) (Impl (Box f) (Box g)))
  | a_t (f : form) : ax_s5 G (Impl (Box f) f)
  | a_4 (f : form) : ax_s5 G (Impl (Box f) (Box (Box f)))
  | a_b (f : form) : ax_s5 G (Impl f (Box (Diamond f)))
  | mp (f g : form) : ax_s5 G (Impl f g) → ax_s5 G f → ax_s5 G g
  | nec (f : form) : ax_s5 empty_set f → ax_s5 G (Box f).
```

In Coq, we define the proof system as an inductive proposition. Firstly, similarly to sets, the type of the proof system is a function of a formula to a proposition. Hence, we can think of the proof system as a property of the formula. The advantage of using inductive proposition [13] is that when making a proof, we will be able to apply each constructor to prove propositions that match the given case, which is exactly what we need from our axiomatic proof system. Secondly, as the argument, we use a set of formulas. The reasoning behind this is that we do not want to use the proof system only to derive theorems of S5, but also to deduce other formulas given a set of formulas. We wish to use any formula in this set as a baseline of the proof, hence we define an additional set membership inference rule (a_0), which says that any formula in a set of formulas is provable from this set.

Lastly, for each axiom and inference rule, we define a constructor with one or more arguments, depending on how many formulas are needed in the given axiom or inference rule. In the definition of each axiom, we use a constructor of the formula type. In order to define the inference rules in Coq level logic, we use Coq level operator implication. In contrast to axioms, these rules can be applied to the proof goals matching the right side of the implication, however, we will then need to prove the left side afterward. Most of the definitions of the individual axioms are not that interesting - they follow directly from the mathematical definition above. However, there is a difference in the definition of the necessitation rule. As the basis of the inference rule, we use an empty set instead of a set of formulas as in other cases. In this way, only the tautologies can be necessitated. If this was not the case, the deduction theorem would not be valid in our system, however, we will need it as part of the completeness proof.

Definition 3.11. *A formula α is deducible (provable) from the set of formulas Γ if only if $\Gamma \vdash \alpha$.*

```
ax_s5 G f
```

In the Coq definition, we simply use the proof with a formula.

Next, we prove the deduction theorem, however, first we need an additional lemma. The lemma states that if a formula can be proven from a set of formulas and this set is a subset of another set, then the formula can be proven from another set too.

Lemma 3.12. $\Gamma \vdash \alpha \wedge \Gamma \subseteq \Delta \Rightarrow \Delta \vdash \alpha$

```
Lemma ax_s5_subset (F G : form_set) (f : form) :
  ax_s5 F f → subset F G → ax_s5 G f.
```

Proof. We prove this lemma by induction on the proof. All the inductive cases can be proven in a straight-forward way. ☐

Theorem 3.13. $\Gamma \vdash (\alpha \to \beta) \iff \Gamma \cup \{\alpha\} \vdash \beta$

```
Lemma deduction_theorem (G : form_set) (f g : form) :
  ax_s5 G (Impl f g) ↔ ax_s5 (add_singleton G f) g.
```

Proof. To prove a biconditional proposition we need to prove the implication in both left-to-right and right-to-left directions. For the left-to-right direction, by Lemma 3.12 we have that $\Gamma \cup \{\alpha\} \vdash (\alpha \to \beta)$. By definition we have $\Gamma \cup \{\alpha\} \vdash \alpha$. Hence, we can use modus ponens to derive $\Gamma \cup \{\alpha\} \vdash \beta$.

For the right-to-left direction, we use proof by induction on the proof. None of the cases is particularly interesting, hence we do not have any other remarks. ☐

Having defined the proof system, we can show how it works. As an example, we present a formal proof of self implication of a formula.

Example 3.14. $\vdash (\alpha \to \alpha)$

Proof.

1. $(\alpha \to ((\beta \to \alpha) \to \alpha)) \to ((\alpha \to (\beta \to \alpha)) \to (\alpha \to \alpha))$ (Axiom 2)
2. $\alpha \to ((\beta \to \alpha) \to \alpha)$ (Axiom 1)

3. $(\alpha \to (\beta \to \alpha)) \to (\alpha \to \alpha)$ (From 1 and 2 by modus ponens)
4. $\alpha \to (\beta \to \alpha)$ (Axiom 1)
5. $\alpha \to \alpha$ (From 3 and 4 by modus ponens)

□

The proof is quite simple. First, we instantiate 1 and 2 with axioms 2 and 1 respectively. From these two propositions, we can deduce 3 by modus ponens. Then we instantiate 4 with axiom 1. Lastly, we once again apply modus ponens to 3 and 4 to deduce 5, which is our goal.

```
Lemma ax_s5_self_impl (G : form_set) (f : form) :
  ax_s5 G (Impl f f).
Proof.
  eapply mp.
  – eapply mp.
    + apply a_2.
    + apply a_1.
  – apply a_1 with (g := f).
Qed.
```

In Coq, we prove this theorem in a similar manner. The main difference is that in the mathematical proof we applied forward reasoning. Contrary, in Coq, we apply backward reasoning. Hence, we first apply the modus ponens rule. As this inference rule requires two assumptions, both of them have to be proven. To indicate these two goals, we use - (minus) and + (plus) symbols. We can relate the first minus part in our Coq proof to part of the formal proof 1-3. The second minus corresponds to assumption 4.

4 Soundness

Soundness is an essential property of any proof system. It states that everything that is provable is true.

Theorem 4.1. $\Gamma \vdash \alpha \Rightarrow (\forall \mathfrak{M}, w : (\forall \beta : \beta \in \Gamma \Rightarrow \mathfrak{M}, w \models \beta) \Rightarrow \mathfrak{M}, w \models \alpha)$

```
Axiom excluded_middle : forall (P : Prop), P ∨ ~P.

Theorem soundness (G : form_set) (f : form) :
  ax_s5 G f →
  forall m w, (forall g, G g → interpret g m w) → interpret f m w.
```

Before proving the soundness of S5, we shall first have a look at the proposition itself. The proposition consists of an implication. The left operand is as expected - α is provable from Γ. On the right side, we have an additional implication. The right side of this implication is what we expect - α is true in w. The left operand is a condition for the model, stating that set membership implies truth. It is necessary to have this assumption in the proposition, otherwise, we would not be able to prove the set membership inference rule, as defined in subsection 3.3.

In order to properly set up the induction in Coq, it is necessary to define the forall statements for model and world after the assumption of the proof. If we defined it over the whole proposition, we would be proving soundness for a particular model and a world, which would not work.

Proof. It is relatively easy to prove the soundness of our proof system. We use a proof by induction on the proof. All we need to do is to show that each axiom is a tautology and that all the inference rules preserve validity. As mentioned earlier, in order to prove the set membership inference rule we need to use the additional assumption we defined. An interesting case is axiom 3. To prove this axiom, we need to use the law of excluded middle. However, Coq is defined using constructive logic, hence there is no law of excluded middle present. To solve this issue we define an additional axiom - not in our proof system but directly in Coq logic. The remaining interesting cases include the axioms T, 4, and B. To prove these cases, we need to use properties of the accessibility relation in the model - reflexivity, transitivity, and symmetry respectively. □

5 Completeness

The second important property we want to prove about our proof system is completeness. It states that everything that is true is provable. Compared to proof of soundness, the completeness proof is much more elaborate. In order to prove it, we first need to define notions of consistent sets, maximal consistent sets, and the canonical model and prove many lemmas that follow from these definitions.

We will prove the completeness with a proof by contrapositive. We will assume that a formula α is not provable from a set of formulas. Then, we will construct a world from this set and $\neg\alpha$. We will show that this is a world of the canonical model and $\neg\alpha$ is true in this world for any set of formulas and α.

5.1 Consistent Sets

In this and the following subsection, we base our definitions on Chellas [4].

Definition 5.1. *Set Γ is consistent ($con(\Gamma)$) if only if $\Gamma \nvdash \bot$*

```
Definition consistent (G : form_set) : Prop :=
  ~(ax_s5 G F_).
```

The Coq definition directly follows from the mathematical definition. Based on this definition we define several theorems.

Theorem 5.2. *$con(\Gamma) \Rightarrow$ not both $\Gamma \vdash \alpha$ and $\Gamma \vdash \neg\alpha$*

```
Lemma consistent_xor (G : form_set) (f : form) :
  consistent G → ~(ax_s5 G f ∧ ax_s5 G (Neg f)).
```

Proof. We prove this by contradiction. $\Gamma \vdash \bot$ follows by modus ponens from $\Gamma \vdash \alpha$ and $\Gamma \vdash \neg\alpha$. By definition of a consistent set, Γ cannot be consistent. □

Theorem 5.3. *$con(\Gamma)$ and $\alpha \in \Gamma \Rightarrow \alpha \notin \Gamma$*

```
Lemma consistent_member (G : form_set) (f : form) :
  consistent G → G f → ~G (Neg f).
```

Proof. We prove this theorem using the previous one. In addition, we use the set membership inference rule. □

Theorem 5.4. *$con(\Gamma)$ and $\Delta \subseteq \Gamma \Rightarrow con(\Delta)$*

```
Lemma consistent_subset (F G : form_set) :
  consistent G → subset F G → consistent F.
```

Proof. We prove this theorem by contradiction, which we achieve using Lemma 3.12. □

Theorem 5.5. *$con(\Gamma)$ and not $con(\Gamma \cup \{\alpha\}) \Rightarrow con(\Gamma \cup \{\neg\alpha\})$*

```
Lemma inconsistent_consistent (G : form_set) (f : form) :
  consistent G → ~consistent (add_singleton G f) → consistent (add_singleton G (Neg f)).
```

Proof. By the law of excluded middle we have either $\Gamma \cup \{\alpha\} \vdash \bot$ or not $\Gamma \cup \{\alpha\} \vdash \bot$. As our assumption is not $con(\Gamma \cup \{\alpha\})$, the right option leads to contradiction. Hence, we assume the left operand. Our next goal is to show that $con(\Gamma)$ leads to contradiction. We achieved this by modus ponens, Theorem 3.13, and double negation elimination. □

We can prove many other theorems from the definition of a consistent set, however, as we do not need them for the completeness proof we do not state them.

5.2 Maximal Consistent Sets

Definition 5.6. *Set Γ is a maximal consistent set ($mcs(\Gamma)$) if only if it is consistent and* $(\forall\alpha : \alpha \in \Gamma$ *or* $\neg\alpha \in \Gamma)$

```
Definition max_consistent (G : form_set) : Prop :=
  consistent G ∧ forall f, (G f ∨ G (Neg f)).
```

Theorem 5.7. $mcs(\Gamma) \Rightarrow (\alpha \in \Gamma \iff \Gamma \vdash \alpha)$

```
Lemma max_consistent_member (G : form_set) (f : form) :
  max_consistent G → (G f ↔ ax_s5 G f).
```

Proof. We prove the left-to-right direction directly using the definition of the set membership inference rule. For the right-to-left direction, by definition of a maximal consistent set we have that $\alpha \in \Gamma$ or $\neg\alpha \in \Gamma$. If the left operand is true, we have proof directly. If the right operand is true, we use proof by contradiction using Theorem 5.2 and a set membership inference rule. □

We define a maximal consistent set as a set that is consistent and for all formulas, the set contains either the formula or its negation. From this definition we can deduce an important property - a formula is provable from a maximal consistent set if only if it is in the set.

Our next goal is to show that any consistent set of formulas can be extended into a maximally consistent set. This theorem is also known as Lindenbaum's lemma [4]. To prove this theorem, we first need to introduce many new definitions and lemmas. First, we assume we have an enumeration of all possible formulas, such that for each formula there is a natural number.

Definition 5.8.

```
Class Countable A `{EqDecision A} := {
  encode : A → positive;
  decode : positive →option A;
  decode_encode x : decode (encode x) = Some x
}.
```

In order to be able to enumerate the formulas in Coq, we show that formula is a member of the type class Countable from package stdpp shown above. To achieve this we define two functions and a proof. The functions convert a formula to and from a tree, such that the tree for different formulas is different too. Secondly, we need proof that if we apply these two functions to a formula we get the formula again. This can be easily shown by induction on a formula.

Having established that formula is a countable type, our goal is to define a recursive function of a set Γ and a natural number n that returns a consistent set containing all the elements of Γ and all the formulas, or their negations, enumerated by the numbers smaller than n. First, we define a helper function insert. Given a set Γ and a natural number, it returns the union of Γ and a singleton set containing the formula decoded from that natural number or the negation of the formula, depending on whether the formula is consistent with Γ or not. We assume that $\alpha = decode(n)$, where function $decode(n)$ returns n^{th} formula of the enumeration.

Definition 5.9.

$$\begin{aligned} insert(\Gamma, n) &= \Gamma \cup \{\alpha\} \text{ if } con(\Gamma \cup \{\alpha\}) \\ &= \Gamma \cup \{\neg\alpha\} \text{ otherwise} \end{aligned}$$

```
Definition insert (G : form_set) (n : nat) : form_set :=
  fun x ⇒
    match decode (Pos.of_nat n) with
    | Some f ⇒
        G x ∨ (consistent (add_singleton G f) ∧ x = f) ∨
        (∼ consistent (add_singleton G f) ∧ x = Neg f)
    | None ⇒ G x
    end.
```

The Coq definition is more complex than the mathematical one. Firstly, there are natural numbers for which there is no formula, therefore we have to use option pattern matching overhead. For the case None, we simply return the initial set - nothing is added to the set. Otherwise, we return a disjunction of three operands. The idea behind this case is similar to the one behind add_singleton function defined in subsection 2.2, with the difference being that here we have one additional disjunct. The second and the third disjunct come with a condition (consistent statement). By the law of excluded middle, we know that one of the two consistent propositions has to be true, hence when making a proof we will be always able to use one of the disjuncts. It might be more intuitive to define it with an if statement for consistent statements instead of as a disjunction of two conjunctions, however, that is not possible. The reason is that we are working with propositions, for which the if statement is not defined.

Based on this function we can prove two lemmas. The first one says that the insert function preserves consistency. The second one states that given a set and any insert into that set, we have a subset of these two sets.

Lemma 5.10. $con(\Gamma) \Rightarrow con(insert(\Gamma, n))$

```
Lemma insert_consistent (G : form_set) (n : nat) :
  consistent G → consistent (insert G n).
```

Proof. We use proof by cases for the law of excluded middle of $con(\Gamma \cup \{\alpha\})$. If it is the negation case, then by Theorem 5.5 we have $con(\Gamma \cup \{\neg\alpha\})$. Now, in both branches of the proof we want to use Theorem 5.4 with the two previously mentioned propositions, hence, what is left to prove is that these two propositions are subsets of $con(insert(\Gamma, n))$. □

Lemma 5.11. $\Gamma \subseteq insert(\Gamma, n)$

```
Lemma insert_subset (G : form_set) (n : nat) :
  subset G (insert G n).
```

Proof. As both cases of the pattern match in insert include the set itself as one of the disjuncts, this is a trivial proof. □

Definition 5.12.

$$step(\Gamma, 0) = \Gamma$$
$$step(\Gamma, n) = insert(step(\Gamma, n-1), n) \text{ for } n > 0$$

```
Fixpoint step (G : form_set) (n : nat) :  form_set :=
  match n with
  | 0 ⇒ G
  | S n ⇒ insert (step G n) n
  end.
```

We proceed to define the step function, which is the function we wanted to define. Again, it is a function of a set and a natural number, returning a set of formulas. This function uses recursively insert by decrementing n, until the base case is reached. Then the set itself is returned. We will use this function to build a maximal consistent set. There are three lemmas related to this definition. The first one is about preserving consistency. The second one states subset property. In this lemma, i and j are natural numbers. The third one states the maximal property.

Lemma 5.13. $con(\Gamma) \Rightarrow \forall n : con(step(\Gamma, n)$

```
Lemma step_consistent (G : form_set) :
  consistent G → forall n, consistent (step G n).
```

Proof. We prove this lemma by induction on n. The base case follows directly. We prove the inductive case by Theorem 5.10. □

Lemma 5.14. $i <= j \Rightarrow \mathit{step}(\Gamma, i) \subseteq \mathit{step}(\Gamma, j)$

```
Lemma step_subset (G : form_set) (i j : nat) :
  i <= j → subset (step G i) (step G j).
```

Proof. We prove this lemma by induction on $i \leq j$. The base case follows directly from the definition. We prove the inductive case using Theorem 5.11. □

Lemma 5.15. $\mathit{con}(\Gamma) \Rightarrow (\alpha \in \mathit{step}(\Gamma, \mathit{encode}(\alpha) + 1)$ *or* $\neg\alpha \in \mathit{step}(\Gamma, \mathit{encode}(\alpha) + 1))$

```
Lemma step_maximal (G : form_set) (f : form) :
  consistent G → step G (S (Pos.to_nat (encode f))) f ∨ step G (S (Pos.to_nat (encode f))) (Neg f).
```

Proof. We prove this lemma we use proof by cases for the law of excluded middle of $\mathit{con}(\mathit{step}(\Gamma, n) \cup \{\alpha\})$. Following many simplifications and rewriting, we will be able to prove both cases by definition without additional lemmas. □

Definition 5.16. *A maximal consistent completion of the set* Γ *(*$\mathit{mcs\_compl}(\Gamma)$*) is a set such that* $\{\alpha \mid \exists i : \alpha \in \mathit{step}(\Gamma, i)\}$

```
Definition max_consistent_set (G : form_set) : form_set :=
  fun f ⇒ exists i, (step G i) f.
```

We proceed to define the completion of a maximal consistent set. It is a function that given a set returns its maximal consistent completion. We define it as a set given by a membership property. The property states that a formula is a member of the set if only if there exists a natural number i, such that $\alpha \in \mathit{step}(\Gamma, i)$.

Finally, we can define Lindenbaum's lemma, which states that if a formula is derivable from maximal consistent completion of a set, then there exists a natural number i, such that the set given by $\mathit{step}(\Gamma, i)$ proves the formula.

Theorem 5.17. $\mathit{mcs\_compl}(\Gamma) \vdash \alpha \Rightarrow \exists i : \mathit{step}(\Gamma, i) \vdash \alpha$

```
Lemma lindenbaum_lemma (G : form_set) (f : form) :
  ax_s5 (max_consistent_set G) f → exists i, ax_s5 (step G i) f.
```

Proof. We prove Lindenbaum's lemma by induction on the hypothesis. We can prove all the cases except modus ponens easily by the definitions. For modus ponens, after the simplification we are left with two inductive hypotheses - $\exists i : \mathit{step}(\Gamma, i) \vdash (\alpha \to \beta)$ and $\exists j : \mathit{step}(\Gamma, j) \vdash \alpha$ - and we want to proof $\exists k : \mathit{step}(\Gamma, k) \vdash \beta$. The problem we face is that we do not know the relation between i and j. Hence, we construct two cases, such that we assume $i \leq j$ in the first case and $j \leq i$ in the second one. Now we are able to prove both cases using modus ponens, Theorem 3.12 and Lemma 5.14. □

Next, we show that our maximal consistent set completion indeed returns a maximal consistent set, supposing the set of formulas is consistent.

Theorem 5.18. $\mathit{con}(\Gamma) \Rightarrow \mathit{mcs}(\mathit{mcs\_compl}(\Gamma))$

```
Lemma max_consistent_set_correct (G : form_set) :
  consistent G → max_consistent (max_consistent_set G).
```

Proof. We divide the proof into two parts - we need to show both that the completion is consistent and that it is maximal. We show the consistency with a proof by contradiction by applying Theorem 5.17 and Lemma 5.13. We show maximality using Lemma 5.15. □

Lastly, we prove a lemma stating that each set is a subset of its maximal consistent completion.

Lemma 5.19. $\Gamma \subseteq \mathit{mcs\_compl}(\Gamma)$

```
Lemma max_consistent_subset (G : form_set) :
  subset G (max_consistent_set G).
```

Proof. We prove this lemma by simplification. □

5.3 Canonical Model

The last concept we need to introduce for the completeness proof is the canonical model for S5 and to prove the theorems about it. We base the definitions in this section on Blackburn [3]. The main idea behind the canonical model is that any consistent set of formulas is true at some point in the model [3]. We proceed with the step-by-step construction of the model by defining each of its components.

Definition 5.20. *The set of worlds W_C of the canonical model is the set of all maximal consistent sets.*

```
Record can_world := {
  world_set :> form → Prop;
  world_set_mcs : max_consistent world_set;
}.
```

As we stated in subsection 3.2, in the model of S5 we do not define the set of all worlds explicitly but define a world just as a type, which is then used in the accessibility relation and the valuation function. This allows us to define a world in the canonical model as a tuple of a set of formulas and a proposition proving that this set is maximally consistent. Using the membership property of maximal consistent set, we define a useful lemma about derivations in the canonical worlds.

Lemma 5.21. $(w \vdash \alpha \Rightarrow w \vdash \beta) \Rightarrow (\alpha \in w \Rightarrow \beta \in w)$

```
Lemma world_closed_derv (G : can_world) (f g : form) :
  (ax_s5 G f → ax_s5 G g) → G f → G g.
```

Proof. We proof this lemma using Theorem 5.7. □

Definition 5.22. (v, w) *is a member of the accessibility relation* R_C *of the canonical model if only if* $\forall \alpha : \alpha \in w \Rightarrow \Diamond\alpha \in v$

```
Definition can_rel (F G : can_world) : Prop :=
  forall x, G x → F (Diamond x).
```

We define the canonical model relation in terms of possibility. We apply backward reasoning. If $R(v, w)$ and the formula is in w, then there must be a possibility of the formula in v. However, we want to define the relation in terms of necessity too. Hence, we define the following lemma showing that the definitions in terms of possibility and necessity are equivalent.

Lemma 5.23. $(v, w) \in R_C \iff \forall \alpha : \Box\alpha \in v \Rightarrow \alpha \in w$

```
Lemma can_relation (F G : can_world):
  can_rel F G ↔ (forall f, F (Box f) → G f).
```

Proof. We need to prove the proposition in both directions of the biconditional. First, we prove the left-to-right direction. As v is maximal, we know that $\alpha \in w$ or $\neg\alpha \in w$. In the first case, the proof is finished. We will show that the second case leads to a contradiction. Since $\neg\alpha \in w$, using the hypothesis we have $\Diamond\neg\alpha \in v$. However, we already have that $\Box\alpha \in v$, which by Theorem 5.21 and $(\Gamma \vdash \Box\alpha \Rightarrow \Gamma \vdash \neg\Diamond\neg\alpha)$ leads to $\neg\Diamond\neg\alpha \in v$. Then, we prove the contradiction using Theorem 5.3. The proof for the other direction is similar, but instead of $(\Gamma \vdash \Box\alpha \Rightarrow \Gamma \vdash \neg\Diamond\neg\alpha)$ we use double negation elimination in the proof. □

In order to construct the model of S5, we need to show that its relation is equivalent.

Lemma 5.24. *equivalent*(R_C)

```
Lemma can_rel_equiv :
  equiv _ can_rel.
```

Proof. By the definition of equivalence relation, we need to show that the relation is reflexive, transitive, and symmetric. We prove the reflexivity using Theorem 5.23 and Theorem 5.21 with axiom T. We prove the transitivity using Theorem 5.21 and ($\Gamma \vdash \Diamond\Diamond\alpha \Rightarrow \Gamma \vdash \Diamond\alpha$), which is a theorem of S5 known as 4$\Diamond$ [4]. We can get it by converting boxes into diamonds and simplification in axiom 4. Lastly, we prove the symmetry using Theorem 5.23 and Theorem 5.21 with axiom B. □

Definition 5.25. *The valuation function V_C of the canonical model is a function of a set Γ and a formula α defined by the set membership function $\alpha \in \Gamma$*

```
Definition can_val (G : can_world) (x : var) : Prop :=
  G (Var x).
}.
```

Having defined the worlds as maximal consistent sets, which by Theorem 5.7 prove a formula if only if it is their member, it is intuitive to use the set membership as a valuation function.

Definition 5.26. *The canonical model $\mathfrak{M}_C$ is a tuple of W_C, R_C, and V_C*

```
Definition can_model : model :=
  {| world := can_world;
     rel := can_rel;
     val := can_val;
     eq := can_rel_equiv;
  |}.
```

Our next goal is to prove the truth lemma, which states that a formula is true in a world of the canonical model if only if it is a member of that world. This lemma will be the last lemma we need to prove the completeness of S5. Before we can prove it, we first need to define and prove two auxiliary lemmas.

Lemma 5.27. $\Diamond\alpha \in v \Rightarrow \exists w : (v, w) \in R_C$ *and* $\alpha \in w$

```
Lemma existence_lemma (w1 : can_model) (f : form) :
  w1 (Diamond f) → exists w2, can_rel w1 w2 ∧ w2 f.
```

Proof. In order to prove this lemma, we need to show for one world w that $(v, w) \in R_C$ and $\alpha \in w$. We will prove it for maximal extension of $\{\beta \mid \Box\beta \in v\} \cup \{\alpha\}$, but in order to be able to construct the world using the maximal extension completion, we need to show that the set is consistent. We show it using proof by contradiction. Supposing the set is inconsistent, we can deduce false from it. By Theorem 3.13 we get $\{\beta \mid \Box\beta \in v\} \vdash \neg\alpha$. Next, we show by induction that $\{\beta \mid \Box\beta \in v\} \vdash \neg\alpha \Rightarrow v \vdash \Box\neg\alpha$. Since v is a maximal consistent set, by Theorem 5.7 we get $\Box\neg\alpha \in v$. However, since our hypothesis is $\Diamond\alpha \in v$, this leads to contradiction by Theorem 5.3.

Having shown that $\{\beta \mid \Box\beta \in v\} \cup \{\alpha\}$ is consistent, we finish the proof by showing that for the world w given by the maximal consistent completion of this set we have $(v, w) \in R_C$ and $\alpha \in w$. Both of the conjuncts can be shown by simplification and subset lemmas from subsection 2.2. □

Lemma 5.28. $(\alpha \in w \Rightarrow \beta \in w) \iff (\alpha \to \beta) \in w$

```
Lemma world_impl (G : can_world) (f g : form) :
  (G f → G g) ↔ G (Impl f g).
```

Proof. We need to prove this lemma in both directions. Firstly, we prove the right-to-left direction by Theorem 5.7, modus ponens and Theorem 5.21. The proof for the left-to-right direction is more complicated. Since w is a maximal consistent set, by definition $\beta \in w$ or $\neg\beta \in w$. In the first case, we can prove the goal by axiom 1, modus ponens, and Theorem 5.21. We split the second case into another two cases using the definition of maximal consistent set. Either $\alpha \in w$ or $\neg\alpha \in w$. In the first case, we reach a contradiction using the hypothesis and Theorem 5.3. In the second case, by using axiom 1, modus ponens and Theorem 5.21 we get $(\neg\beta \to \neg\alpha) \in w$. Using axiom 3, modus ponens and Theorem 5.21 we prove the goal $(\alpha \to \beta) \in w$. □

Lemma 5.29. $\forall \alpha, v : \alpha \in v \iff \mathfrak{M}_C, v \models \alpha$

```
Lemma truth_lemma :
  forall (f : form) (w1 : can_model) , w1 f ↔ interpret f can_model w1.
```

Proof. We prove the truth lemma by induction on formula. Since there are four constructors of a formula type, we will have four cases to consider. Furthermore, we can split each case into two cases, since we need to prove a bi-implication. We prove the cases for false and variable constructors by definition in both directions. For the case of implication, we use induction hypotheses and Theorem 5.28 in both directions. For the box case, we prove the left-to-right direction using Theorem 5.23. The right to left direction is the only difficult case. Since v is a maximally consistent set, by definition either $\Box\alpha \in v$ or $\neg\Box\alpha \in v$. In the first case, we directly prove the goal. In the second case, we can derive $\Diamond\neg\alpha \in v$ from $\neg\Box\alpha \in v$. Then, by Theorem 5.27 we have $(v, w) \in R_C$ and $\neg\alpha \in w$. However, by the application of the inductive hypothesis in the hypothesis of this case we have $\alpha \in w$. Since w is consistent we can prove a contradiction by Theorem 5.4. □

5.4 Completeness Proof

We have proven all the necessary lemmas and we are now ready to prove the completeness of our proof system. Similarly to the proposition of soundness, we have an additional assumption that a set membership is valid inference rule. Having defined all the needed lemmas, the proof ends up being not very complex.

Theorem 5.30. $(\forall \mathfrak{M}, w : (\forall \beta : \beta \in \Gamma \Rightarrow \mathfrak{M}, w \models \beta) \Rightarrow \mathfrak{M}, w \models \alpha) \Rightarrow \Gamma \vdash \alpha$

```
Theorem completeness (G : form_set) (f : form) :
  (forall (m: model) (w : m), (forall g, G g → interpret g m w) →
  interpret f m w) → ax_s5 G f.
```

Proof. We prove the completeness using a proof by contradiction. By the law of excluded middle, we have either $\Gamma \vdash \alpha$ or $\Gamma \nvdash \alpha$. The first case is trivial. Supposing $\Gamma \nvdash \alpha$, we will construct a world $w \in W_C$ from the set $\Gamma \cup \{\neg\alpha\}$ using the maximal consistent completion of this set. In order to do that we first need to prove that $\Gamma \cup \{\neg\alpha\}$ is consistent. We prove this by contradiction. Using Theorem 3.13 and double negation elimination, we get $\Gamma \vdash \alpha$, which is a contradiction with $\Gamma \nvdash \alpha$.

By Theorem 5.19 and Theorem 5.29, we have that $\mathfrak{M}_C, w \models \neg\alpha$. If we can show that $\forall \beta : \beta \in \Gamma \Rightarrow \mathfrak{M}_C, w \models \beta$, we will be able to prove $\mathfrak{M}_C, w \models \alpha$ using the hypothesis, which will lead to a contradiction.

We show $\Gamma \cup \{\neg\alpha\} \subseteq w$, by Theorem 5.19 and the subset lemmas from subsection 2.2. Then, again by using these lemmas, we deduce that $\beta \in w$. By Theorem 5.29 we have $\mathfrak{M}_C, w \models \beta$, which finalizes the proof. □

6 Conclusion and Future Work

The goal of this project was to formalize modal logic S5 in Coq proof assistant. We started with the definitions of formula and model. Then, using these definitions we defined validity in a model and a proof system. Our next step was to prove the relationships between these two concepts - soundness and completeness. It was not difficult to prove the soundness of our system. However, in order to prove completeness, we needed to define many other auxiliary concepts and lemmas - that being consistent sets, maximal consistent sets, and the canonical model. For all the definitions we showed both mathematical and Coq definitions, which we justified with our design decisions. Having proven both the soundness and the completeness of our system, it is ready to be used for making proofs in modal logic S5.

The most challenging part of this project was working with Coq as we had not had any experience in it before the start of the project. Despite having read a textbook [13] on Coq and other Coq materials [11] before our start, we often run into Coq-related problems. Furthermore, once we run into a such problem, we found it difficult to find the solution, especially compared to more popular programming languages, which lead us to spend much time on issues not directly related to formalization. Nevertheless, as we were working on the project we became more confident in Coq, and the time to solve these problems decreased. If we were to begin this project once again, we would first attempt a smaller-scale project to get some extra experience with Coq before attempting the whole formalization.

There are two clear paths for future work. Firstly, this project could be relatively easily extended to prove all modal logics, whose accessibility relations have only some of the properties of S5's relation and no other additional properties. That includes modal logics K, T, S4, B, etc. This can be achieved by removing the axioms, the cases for these axioms in the inductive proofs, and the properties not needed for these logics. Since we defined the S5 proof system with three axioms instead of two - each one for one of its properties - no other changes would be needed. Secondly, we could define an entirely different type of proof system, such as the tableau method, and prove its soundness and completeness. This would be useful since the Hilbert system is not the most intuitive system to work with. However, compared to the previous extension, both soundness and completeness proofs would have to be proven from scratch.

7 Acknowledgements

I would like to thank my supervisors Dan Frumin and Revantha Ramanayake for their assistance with the project. I want to thank Dan for our weekly meetings, where he showed me various effective ways to use Coq and our discussions about the proofs. I want to thank Revantha for his helpful feedback on the presentation and thesis and the general guidance of the project.

References

[1] Kaja Bednarska and Andrzej Indrzejczak. Hypersequent Calculi for S5: The Methods of Cut Elimination. *Logic and Logical Philosophy*, 24(3):277–311, Aug. 2015.

[2] Bruno Bentzen. A Henkin-Style Completeness Proof for the Modal Logic S5. In Pietro Baroni, Christoph Benzmüller, and Yi N. Wáng, editors, *Logic and Argumentation*, pages 459–467, Cham, 2021. Springer International Publishing.

[3] P. Blackburn, M. de Rijke, and Y. Venema. *Modal Logic*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2002.

[4] Brian F. Chellas. *Introduction*, page 3–24. Cambridge University Press, 1980.

[5] Thierry Coquand and Gérard Huet. Constructions: A higher order proof system for mechanizing mathematics. In Bruno Buchberger, editor, *EUROCAL '85*, pages 151–184, Berlin, Heidelberg, 1985. Springer Berlin Heidelberg.

[6] Christian Doczkal and Gert Smolka. Constructive Formalization of Classical Modal Logic. 2011.

[7] Melvin Fitting. A simple propositional s5 tableau system. *Annals of Pure and Applied Logic*, 96(1):107–115, 1999.

[8] Yasuyuki Imai and Kiyoshi Iséki. On axiom systems of propositional calculi, I. *Proceedings of the Japan Academy*, 41(6):436 – 439, 1965.

[9] Herman Ruge Jervell. *Modal Logic*. Logos Verlag Berlin, 2013.

[10] Stephen Cole Kleene. *Introduction to Metamathematics*. North-Holland Publishing Company, 1952.

[11] Christine Paulin-Mohring. *Introduction to the Coq Proof-Assistant for Practical Software Verification*, pages 45–95. Springer Berlin Heidelberg, Berlin, Heidelberg, 2012.

[12] Michiel Philipse. Distributed Knowledge Proofs in the Coq Proof Assistant, 2021.

[13] Benjamin C. Pierce, Arthur Azevedo de Amorim, Chris Casinghino, Marco Gaboardi, Michael Greenberg, Cătălin Hriţcu, Vilhelm Sjöberg, and Brent Yorgey. *Logical Foundations*, volume 1 of *Software Foundations*. Electronic textbook, 2021. Version 6.1.

[14] The Coq Development Team. The coq proof assistant. https://coq.inria.fr/about-coq, 2022. Accessed: 26.3.2022.

[15] Floris van Doorn. Propositional calculus in coq. *ArXiv*, abs/1503.08744, 2015.

[16] Minchao Wu and Rajeev Goré. Verified Decision Procedures for Modal Logics. In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving (ITP 2019)*, volume 141 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 31:1–31:19, Dagstuhl, Germany, 2019. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.

A Source Code

File	Content
prop.v	Axiom excluded middle and lemmas about propositions
nat.v	Lemmas about natural numbers
set.v	Definition of set and lemmas about it
form.v	Definition of formula and set of formulas
model.v	Kripke frame for S5, lemmas about its properties and definition of validity in model
proof.v	Definition of the proof system
soundness.v	Proof of soundness
inference.v	Various inference lemmas in the proof system that does not make use of modal axioms
inference_modal.v	Various inference lemmas in the proof system that does make use of modal axioms
deduce.v	Subset lemma for inference in the proof system and the deduction theorem
consistent.v	Definition of consistent set of formulas and lemmas about it
encode.v	Proof that formula is a countable type
maximal.v	Definition and construction of maximal consistent set and lemmas about it
canonical.v	Definition of canonical model and lemmas about it
completeness.v	Proof of completeness

Table 1: Table showing content of each file. The files are in order of compilation.