# A Formalization of Dynamic Epistemic Logic

Carnegie Mellon University

Paula Neeley

April 14, 2021

# Abstract

In this thesis, I describe a formalization of various modal logics in the Lean theorem prover that includes both model and proof theoretic results. First, I address the formalization of model theoretic results in frame definability and undefinability, as well as proof theoretic results in soundness and completeness for the modal systems K, T, S4, S5. Next, I address the formalization of soundness and completeness for the epistemic systems S5 and PA (public announcement logic without common knowledge). To the best of my knowledge, this thesis represents the first formalization of modal model theoretic arguments and constructions beyond completeness in Lean, and the first soundness and completeness results for public announcement logic in any theorem prover to date. This thesis can also serve as a basis for further formalization work in modal logic in Lean – such as the formalization of topological semantics or other epistemic systems of knowledge and belief.

# Chapter 1

# Introduction

> The distribution of information among autonomous agents, the transferral of information between agents, and the gain and loss of information by agents over time are critical characteristics of many environments. This is particularly true of environments where computers are applied, since the control and distribution of information is a primary function of computer systems and a central issue in computer science. It can be very valuable in analyzing any such environment to be able to represent explicitly and to reason about the state of information and the dynamics of information.
>
> (Davis and Morgenstern [8])

In an effort to keep up with the newly developed Airbus A320 model, rival manufacturer Boeing upgraded the engine on its popular 737 model in 2014. This upgrade made the engine larger, and it no longer fit under the wing in keeping with the plane's current design. Boeing's solution was to move the engine on the wing so that it would be slightly higher on the 737. Boeing called this model the 737 MAX.

Moving the engine on the 737 had a side effect, however. The center of gravity shifted, and when the 737 MAX was in full thrust – such as during takeoff – the nose would tend to point too far upward, leading to the possibility of stalls. Boeing came up with a workaround: they installed software that automatically pushed the nose downward if the pilot flew the plane at too high of an angle. Boeing called this the *Maneuvering Characteristics Augmentation System*, or MCAS.

In 2018, several American pilots complained to the federal government that the 737 MAX was "nosing down." On October 29, 2018, Lion Air Flight 610 took off from Jakarta, Indonesia in a 737 MAX model. In the flight report, which shows the plane's altitude over time, one can see that the nose of the plane continually lurched downward and the plane struggled to gain altitude. Reports show that falsely high angle of attack (AoA) inputs were triggering MCAS, and only 12 minutes after take off the plane crashed into the Java Sea, killing 189 people. Ethiopian Airlines Flight 302 also crashed on March 10, 2019 for essentially the same reason, killing 157 people.

In safety critical systems such as these, *formal methods* can be instrumental in preventing such catastrophic outcomes. Formal methods describes the use of rigorously specified mathematical models and mathematical proofs to ensure correct behavior in the design and implementation of hardware, software, and autonomous systems. Mathematical proofs of

correctness provide an additional level of safety-checks beyond what is possible with mere testing.

Within formal methods – and interactive theorem proving in particular – epistemic modal logics of knowledge and belief can play a role in the verification of subsystems that transmit or communicate information within a larger system. Think of it this way: in the same sense that the computer scientist aims to discern how and when subsystems can avoid and detect errors in order to continue functioning appropriately even under abnormal conditions within a larger system, the epistemologist aims to discern how and when an agent can be said to know some piece of information even when he or others in his group that he communicates with may be in error about related information [15].

To develop methods of detecting and avoiding errors in both types of problems, it is advantageous to formally define epistemic notions in terms of logic and to formally distinguish something like knowledge from weaker concepts such as belief. Indeed, computer systems often fail to behave as they were designed, and in these cases the subsystems that transfer information within a larger system may be subject to the types of errors that are analogous to false beliefs [15]. For instance, in the case of the 737 MAX, the problem was not that the plane actually had an excessively high angle of attack, the problem was that inputs communicated to MCAS from various subsystems gave MCAS reason to in some sense believe that the angle was elevated and that the plane needed to nose down. Appropriately modeling computer systems as agents communicating within a group and making use of formalizations of epistemic modal logics as verification tools is one possible way to verify that mistakes like these can not happen.

However, formalizations of logic and mathematics have theoretical implications beyond their applications in software verification. In 2020, Kevin Hartnett at Quanta Magazine published a piece called *Building the Mathematical Library of the Future* [10]. In it, the author describes the growing community of Lean users devoted to the project of digitizing mathematics. The expected benefits of digital mathematics, Hartnett writes, range from mundane uses such as computers grading students' math homework, reviewing journal submissions in order to find errors that human reviewers might miss, and filling in the tedious technical details of a proof, to more transcendent uses – such as utilizing machine learning and artificial intelligence to eventually discover and prove new mathematics [10].

Indeed, if computers were to ever become proficient at proving new theorems in mathematics or logic, it would be through an approach that employs machine learning paradigms, and libraries of formalized mathematics would be fundamental for supplying the training sets to learning algorithms. The mere reliance on mathematical definitions and the statement of theorems for training sets would never suffice; learning algorithms need the sort of training data that supplies every single detail in the proof of a theorem, and massive amounts of such data, at that. But that is *exactly* what libraries of formalized mathematics, such as Lean's mathlib, provide. Indeed, Google made waves in 2019 when its deep reinforcement learning based automated theorem prover, DeepHOL, proved over 1200 mathematical theorems using training data extracted from the theorem prover HOL Light [7]. What is more, a group of researchers have recently started using Lean for machine learning with the creation of LeanStep, a dataset and environment for neural theorem proving.

There is much work to be done on the part of working mathematicians and logicians to grow the libraries of digital mathematics, since many of the most common theorem provers

such as Coq, Lean, Isabelle, or HOL only contain as much mathematics as an undergrad would encounter in his or her baccalaureate education. Moreover, very few formalizations of modal or epistemic logics exist within these most common proof assistants. This gap is what I aim to address in this thesis.

To that end, this thesis describes a novel formalization of modal and epistemic logics within the Lean theorem prover partially motivated by the textbook *Dynamic Epistemic Logic* by van Ditmarsch, van der Hoek, and Kooi [16]. In particular, in this thesis I present:

- A deep embedding of public announcement logic without common knowledge in Lean,
- Proof theoretic contributions, including soundness and completeness results for the modal proof systems K, T, S4, S5, as well as the S5 proof system of epistemic logic and the PA proof system of public announcement logic without common knowledge,
- Model theoretic contributions, including frame definability and undefinability results via invariance under disjoint unions, generated subframes, bisimulations, and surjective bounded morphisms.

In chapter 2, I outline the theoretical foundations of modal and dynamic epistemic logic. Chapter 3 describes the formalization of these logical theories in Lean. Chapter 4 provides a discussion of related work, and chapter 5 addresses ideas for future work and conclusions.

# Contents

# Chapter 2

# Preliminaries

This chapter describes the systems of propositional modal and dynamic epistemic logic in detail. Section 2.1 covers the theory of modal logic, which will provide a foundation for later explicating the theory of dynamic epistemic logic in section 2.2. Throughout this chapter, I assume the reader is familiar with classical propositional and first-order logic.

## 2.1 Modal Logic

Propositional modal logic is a type of logic used to qualify the truth of a proposition. For instance, the statement, "Alice is late" can be qualified by saying, "Alice is *always* late," where the modal *always* functions as a qualifier to the sentence.

Modal logic was first developed by Aristotle in his modal syllogistic (in book I of the *Prior Analytics*) and was further studied in both the Hellenistic and Medieval periods. However, its modern development began with the 1910 Harvard dissertation of C.I. Lewis [11]. Our discussion of modern modal logic will begin with the syntax and semantics, followed by model theoretic frame definability and undefinability results, and finally proof theoretic soundness and completeness results.

### 2.1.1 Syntax and Semantics

In the language of modal logic, truth qualifiers (such as *always*, mentioned above) are represented by adjoining special operators to propositional formulas. The most commonly used operator is called the "necessity operator" or "box", and it is usually denoted $\Box$. We can thus characterize the propositional modal language with the following definition.

**Definition 2.1.1 (The Language of Modal Logic)** The well-formed formulas $\varphi$ of the propositional modal language are given recursively by the rules:

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \psi \mid \Box\varphi$$

where $p \in \text{PROP}$ ranges over a countable set of primitive propositions. In the discussion that follows, this language is denoted $\mathcal{L}$.

Traditionally, unary operators such as "box" were interpreted in an alethic sense, with $\Box\varphi$ expressing "it is logically necessary that $\varphi$" and its dual, $\Diamond\varphi$ (an abbreviation for $\neg\Box\neg\varphi$) expressing "it is logically possible that $\varphi$." However, the alethic interpretation is just one of many, and modal logic has a variety of applications due to the fact that necessity and possibility have a variety of interpretations. For instance, necessity can refer to epistemic necessity (e.g., what must be the case according to what I know), doxastic necessity (...according to what I believe), temporal necessity (...according to this point in time), or deontic necessity (...according to my moral duties).[1]

All of these interpretations have in common the fact that they can be mathematically evaluated in terms of *bounded universal quantification*. For example, our earlier statement that "Alice is always late" can be thought of as expressing the fact that, in all possible configurations of the world, Alice is late.

Such *possible world* semantics are a development originally due to Rudolf Carnap and further refined by Saul Kripke. A possible world is a complete and consistent orientation for the domain of discourse, or one way that things could be out of all the possible ways, depending on the context.

By way of illustration, many of us believe that our world as a whole need not have been exactly the way it is. Rather, things might have been different in a countless variety of trivial or nontrivial ways. Our world – the actual world – is just one of many possible worlds from which we can state propositions and ascribe truth values to them.[2]

In practical applications of modal logic, sets of possible worlds might represent states of a computer program, nodes of a distributed system, or configurations of a discrete dynamical system that evolves over time. Sets of accessible worlds might represent the subset of states of the program that follow from performing a certain function, the reachable nodes from a given node in the distributed system, or the next state given a single time step in the dynamical system.

Possible world semantics are formally defined via *Kripke frames*. For this, we have the following definition.

**Definition 2.1.2 (Kripke Frame)** A Kripke frame is a pair $F = (W, R)$ where $W$ denotes a non-empty set of possible worlds and $R \subseteq W \times W$ denotes a binary relation (called the *accessibility relation*) between worlds.

We can write $w_1 R w_2$ to express the fact that world $w_2$ is accessible from world $w_1$, and we can define $R(w_1) = \{w : w_1 R w\}$ to be the set of all worlds accessible from $w_1$.

Kripke frames enable expressions of relational properties, and they are frequently modeled using relational graphs (also called directed graphs) [19]. Graphs are fundamental for representing the behavior and transformations of systems in almost every area of computer science, making modal logic a natural way to reason about such systems.

[1] Sometimes new symbols are introduced to the formal language to clarify which interpretation should be used. For example, $B\varphi$ might be used to express the modal sentence, "It is believed that $\varphi$." Otherwise, the conventional symbols $\Box$ and $\Diamond$ are routinely used when the context is clear.

[2] For example, it may be true in our world that Ann lives in San Francisco. However, it is not *necessarily* true that she lives in San Francisco. Perhaps if Ann took a different job in her field, or worked in a different field, she would live in Seattle instead. In this case, we say that the possible world in which Ann lives in Seattle is accessible from our world, the world in which Ann lives in San Francisco.

Reasoning about what kind of statements hold in a relational model requires a notion of truth, however. This brings us to our next definition.

**Definition 2.1.3 (Kripke Model)** A Kripke model $M = ((W, R), v)$ is a tuple where $(W, R) = F$ denotes a frame and $v : \text{PROP} \to 2^W$ denotes a valuation function over the primitive propositions.[3]

The valuation $v$ specifies the truth values of primitives at various worlds in $W$, so $w_1 \in v(p)$ means that "$p$ is true at world $w_1$." With definition (2.1.3) in hand, we can extend our possible world notion of truth for primitive propositions to all modal formulas recursively.

**Definition 2.1.4 (Validity)** Let $x$ be a world in a model $M = ((W, R), v)$. We say that formula $\varphi$ is valid (alternatively, *true*) at $x$ in $M$ whenever

$$\begin{array}{lll} (M, x) \models p & \text{iff} & x \in v(p) \\ (M, x) \models \neg\varphi & \text{iff} & (M, x) \not\models \varphi \\ (M, x) \models \varphi \wedge \psi & \text{iff} & (M, x) \models \varphi \text{ and } (M, x) \models \psi \\ (M, x) \models \Box\varphi & \text{iff} & \forall y \in W,\ xRy \text{ implies } (M, y) \models \varphi. \end{array}$$

On a related note, we can also say what it means for a formula $\varphi$ to be a semantic consequence of a set of formulas (also called a *context*) $\Gamma$.

**Definition 2.1.5 (Global Semantic Consequence)** A formula $\varphi$ is a global semantic consequence of a context $\Gamma$ if and only if, for any model $M$, if $\Gamma$ is true in $M$ (that is, all formulas $\psi \in \Gamma$ are true in $M$) then $\varphi$ is true in $M$.

In the next section, I will build on these basic definitions to explicate more advanced topics in the model theory.

### 2.1.2 Frame Definability and Undefinability

The definition of local validity given in definition (2.1.4) can be extended to more global contexts. We say that a formula $\varphi$ is valid:

- in a model $M$, if $(M, x) \models \varphi$ for every $x$ in $M$, denoted $M \models \varphi$,
- in a frame $F$, if $(F, v) \models \varphi$ for every valuation $v$, denoted $F \models \varphi$,
- on a class of frames $\mathcal{F}$, if $F \models \varphi$ for every $F \in \mathcal{F}$, denoted $\mathcal{F} \models \varphi$,
- universally, if $\varphi$ is valid on every frame, denoted $\models \varphi$.

One question of interest to modal logicians concerns the connection between an accessibility relation and the truth of specific formulas in frames admitting that relation. For example, consider the reflexive relation where every world is accessible from itself,

$$\forall x \in W, Rxx.$$

[3]We sometimes write $M = (F, v)$ as shorthand for $M = ((W, R), v)$.

It is not difficult to check that the formula $\Box p \to p$ is valid on all frames having the reflexive relation.[4] But does the converse hold? If a frame can be shown to validate this formula, must that frame be reflexive? These questions are the subject of *frame definability*.

**Definition 2.1.6 (Definability)** A formula $\varphi$ defines a class of frames $\mathcal{F}$ if, for every frame $F$ we have that $F \in \mathcal{F}$ if and only if $F \models \varphi$. That is, a modal formula defines a class if all frames within the class satisfy the formula, and all frames outside the class can be found to refute the formula.

For instance, the formula $\Box p \to p$ defines the class of reflexive frames,[5] the formula $\Box p \to \Box\Box p$ defines the class of transitive frames, and the formula $\Box(\Box p \to p) \to \Box p$ (also known as Löb's formula) defines the class of frames that are transitive and admit no infinite $R$-paths.[6]

In modal logic, we often think of frame classes and their associated accessibility relations as capturing properties of directed graphs. So is every graph property definable by a modal formula? As it turns out, the answer is no. There are countably many formulas in the modal language, but an uncountable number of relational properties for graphs. Thus, we say that a class of frames is *undefinable* if there does not exist a modal formula that defines it. Examples of undefinable properties include the class of irreflexive frames, the class of finite frames, and the class of frames in which every point has a predecessor.[7]

Proving undefinability is generally more involved than proving definability, for instead of checking that a single modal formula defines a class of frames, we must verify that no modal formula can define it. To prove such a general result, we rely on methods of invariance under transformations. The idea here is that if validity of formulas ought to be preserved under certain frame transformations, but a class of frames $\mathcal{F}$ is not closed under a transformation, then $\mathcal{F}$ is not modally definable. I shall begin our discussion of undefinability with a few definitions.

**Definition 2.1.7** The theory of $x$ in $M$ is the set of all modal formulas that are true at $x$. That is, $Th_M(x) = \{\varphi : (M, x) \models \varphi\}$.

**Definition 2.1.8 (Modal Equivalence)** We say that $(M, x)$ and $(M', x')$ are modally equivalent if $Th_M(x) = Th_{M'}(x')$.

Undefinability theorems often make use of modal equivalence,[8] but when are $(M, x)$ and $(M', x')$ modally equivalent? Clearly, if we can define an isomorphism $f : M \to M'$ that

[4]Proof sketch: Suppose $F \in \mathcal{F}_{ref}$, so $\forall x \in W, Rxx$. Also suppose $x \models \Box p$. By definition of validity for the necessity operator, $Rxy$ implies $y \models p$. Since $Rxx$, it follows that $x \models p$. Since $F \in \mathcal{F}_{ref}$ is arbitrary, it follows that $\mathcal{F}_{ref} \models \Box p \to p$.

[5]Proof sketch: We have already shown the forward direction, what remains to be shown is if $F \models \Box p \to p$ then $F \in \mathcal{F}_{ref}$. By contrapositive, suppose that $F \notin \mathcal{F}_{ref}$, so $\exists x \in W$ such that $xRx$. Let $v(p) = W \setminus \{x\}$. Then trivially $x \models \Box p$, but $x \not\models p$. It follows that $F \not\models \Box p \to p$.

[6]An $R$-path is a sequence $\langle x_1, x_2, ..., x_n\rangle$ such that for all $i \in \{1, ..., n\}$, $x_1 R x_{i+1}$. An $R$-path may be finite or infinite.

[7]That is, if every world $x$ in the model is reachable from some other world $y$.

[8]To prove undefinability, we assume that some general formula $\varphi$ defines a class, and we pick an arbitrary model and world $(M, x)$ within that class. We then transform the model to $(M', x')$ and show that $Th_M(x) \neq Th_{M'}(x')$, contradicting the original claim.

maps $x$ to $x'$, then $(M, x)$ and $(M', x')$ are modally equivalent. But the condition of being isomorphic is stronger than necessary. On the other hand, homomorphisms are too weak to yield invariance.[9] Thus, I will introduce a series of examples of situations where modal equivalences arise, that will later be pulled together under the common name of *bisimulation* to yield invariance.

**Definition 2.1.9 (Disjoint Union)** Let $M_1 = (X_1, R_1, v_1)$ and $M_2 = (X_2, R_2, v_2)$ be such that $X_1 \cap X_2 = \emptyset$. Then the disjoint union of $M_1$ and $M_2$ is the model $M1 \sqcup M_2 = (X, R, v)$, where

$$\begin{aligned} X &= X_1 \cup X_2, \\ R &= R_1 \cup R_2, \\ v(p) &= v_1(p) \cup v_2(p). \end{aligned}$$

In other words, disjoint unions gather together all the information from two smaller models into one bigger model, but they leave the important aspects of the two smaller models (such as the relationship between points or the valuation of atomics) unchanged.[10]

**Theorem 2.1.10** Modal satisfaction is invariant under disjoint unions. That is, for all $x \in X_1$, $Th_{M_1}(x) = Th_{M_1 \sqcup M_2}(x)$.[11]

**Proof 2.1.10.1** The proof proceeds by induction on $\varphi$. See theorem `invariance_dunion` in appendix A for the full proof in Lean.

Disjoint unions let us in on a secret we otherwise might have missed: modal satisfaction is local in nature. If one evaluates a formula $\varphi$ at world $x$, it should be irrelevant whether that evaluation happens at $x \in M_1$ or $x \in M_1 \sqcup M_2$.

Disjoint unions are useful for transforming smaller models into bigger ones – but to prove a variety of invariance results, we also might need methods for doing the reverse. Indeed, it would be helpful to know what points we could safely remove from a model without affecting validity.

**Definition 2.1.11 (Generated Submodel)** Let $M = (X, R, v)$ and $x \in M$. The generated submodel of $M$ at $x$ is the model

$$M_x = (R^*(x), R|_{R^*(x)}, v|_{R^*(x)}),$$

where $R^*$ is the reflexive, transitive closure of $R$.

In other words, the generated submodel of $M$ at $x$ repeatedly takes all the points that $x$ is related to, and the points those points are related to, potentially ad infinitum, and forms a new model with these points, $M_x$.

[9] From Blackburn's textbook *Modal Logic*, "...although homomorphisms reflect the structure of the source in the structure of the target, they do not reflect the structure of the target back in the source." [6]

[10] We can more generally take the disjoint union of $n$ many models by letting $M_1$ be a disjoint union of $(n-1)$ many models.

[11] By symmetry, the result also holds for all $x \in X_2$.

**Theorem 2.1.12** Modal satisfaction is invariant under generated submodels. That is, for all $y \in R^*(x)$, $Th_{M_x}(y) = Th_M(y)$.

**Proof 2.1.12.1** By induction on $\varphi$. See theorem `invariance_gen_submodel` in appendix A for the full proof in Lean.

Disjoint unions and generated submodels are very specific types of transformations, but our next type of transformation – the bounded morphism – is more general. In particular, it specifies the exact necessary and sufficient conditions for invariance in the spectrum between homomorphic and isomorphic transformations.

**Definition 2.1.13 (Bounded Morphism)** Let $F_1 = (X_1, R_1)$ and $F_2 = (X_2, R_2)$. A bounded morphism (or $p$-morphism) from $F_1$ to $F_2$ is a function $f : X_1 \to X_2$ such that

(FORTH) if $y_1 \in R_1(x_1)$, then $f(y_1) \in R_2(f(x_1))$,

(BACK) if $y_2 \in R_2(f(x_1))$, then there exists a $y_1 \in R_1(x_1)$ such that $f(y_1) = y_2$.

**Theorem 2.1.14** If $f$ is a surjective bounded morphism from $F_1$ to $F_2$, then if $F_2 \not\models \varphi$, then also $F_1 \not\models \varphi$.

**Proof 2.1.14.1** By induction on $\varphi$. See theorem `pull_back` in appendix A for the full proof in Lean.

**Corollary 2.1.14.1** If $\mathcal{F}$ is a class of frames such that $F_1 \in \mathcal{F}$ and $F_2 \notin \mathcal{F}$, then if there exists a surjective, bounded morphism from $F_1$ to $F_2$, then $\mathcal{F}$ is not definable by a formula in the basic modal language.

**Proof 2.1.14.2** See theorem `invariance_pull_back` in appendix A for the full proof in Lean.

Historically, these constructions were used in modal logic for over a decade before the unifying concept of bisimulation was introduced. Bisimulations characterize when two states' atomic valuations and relationship to other points can be matched up. In other words, when two states "look the same" and "act the same." As it turns out, disjoint unions, generated submodels, and surjective bounded morphisms are all examples of bisimulations.

**Definition 2.1.15 (Bisimulation)** Let $M_1 = (X_1, R_1, v_1)$ and $M_2 = (X_2, R_2, v_2)$. A bisimulation between $M_1$ and $M_2$ is a binary relation $\sim \subseteq X_1 \times X_2$ such that for $x_1 \in X_1$ and $x_2 \in X_2$, if $x_1 \sim x_2$, then

(BASE) for all $p \in$ PROP, $x_1 \in v_1(p)$ iff $x_2 \in v_2(p)$,

(FORTH) for all $y_1 \in R_1(x_1)$, there exists a $y_2 \in R_2(x_2)$ such that $y_1 \sim y_2$,

(BACK) for all $y_2 \in R_2(x_2)$, there exists a $y_1 \in R_1(x_1)$ such that $y_1 \sim y_2$.

**Theorem 2.1.16** Modal satisfaction is invariant under bisimulation. That is, if $\sim$ is a bisimulation between $M_1$ and $M_2$ and $x_1 \sim x_2$, then $Th_{M_1}(x_1) = Th_{M_2}(x_2)$.

**Proof 2.1.16.1** By induction on $\varphi$. See theorem `invariance_bisim` in appendix A for the full proof in Lean.

This concludes the discussion of modal definability and undefinability. In the next section, I will discuss modal soundness and completeness results.

### 2.1.3 Soundness and Completeness

So far I have characterized modal logic from a model theoretic perspective. Let us now turn our attention to the proof theoretic perspective via the study of axiomatic systems and deduction. I shall begin by defining the axioms and rules of inference to be added to the classical propositional calculus in order to build a modal calculus. I will follow this discussion with modal soundness and completeness results.

**Definition 2.1.17 (System K)** The modal axiom system $\mathsf{K}$ is comprised of

- all instances of propositional tautologies,
- all instances of the scheme $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$, (DISTRIBUTION)
- from $\varphi$ and $\varphi \to \psi$, we may infer $\psi$, (MODUS PONENS)
- from $\varphi$, we may infer $\Box\varphi$. (NECESSITATION)

The distribution axiom scheme, $\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$ is often referred to itself as $K$. Other well-known axiom schemes are:

$T$: $\Box\varphi \to \varphi$,
$4$: $\Box\varphi \to \Box\Box\varphi$,
$5$: $\Diamond\varphi \to \Box\Diamond\varphi$.

which form the axiom systems (systems in sans-serif, axioms in italics):

$\mathsf{T} := \mathsf{K} + T$,
$\mathsf{S4} := \mathsf{T} + 4$,
$\mathsf{S5} := \mathsf{S4} + 5$.

These axioms (the propositional tautologies, as well as $K$, $T$, $4$, and $5$) taken together with the rules of inference (modus ponens and necessitation) can be used to derive additional formulas via the method of deduction.

**Definition 2.1.18 (Deduction)** Given any axiom system $\mathsf{AX}$, a deduction from $\mathsf{AX}$ is a finite sequence of formulas $\varphi_1, \varphi_2, ..., \varphi_n$ such that, for all $1 \leq i \leq n$, either:

- $\varphi_i$ is an axiom,
- $\varphi_i$ follows from $\varphi_j$, $\varphi_k$ for some $j, k \leq i$ by modus ponens,
- $\varphi_i$ follows from $\varphi_j$ for some $j \leq i$ by necessitation.

We denote a deduction from $\mathsf{AX}$ of $\varphi_n$ by $\vdash_{\mathsf{AX}} \varphi_n$ and say that $\varphi_n$ is provable in system $\mathsf{AX}$.

As is common in all systems of logic, we are concerned with whether the notion of deduction given in definition (2.1.18) coincides with the notion of validity given in definition (2.1.4). That is, whether the formulas we can derive are actually true, and whether we can derive all the true formulas. For this we have the following theorems.

**Theorem 2.1.19 (Soundness)** The axiom system $\mathsf{K}$ over the language $\mathcal{L}$ is sound with respect to the class of all frames. That is, $\vdash_{\mathsf{K}} \varphi$ implies $\models \varphi$.

**Proof 2.1.19.1** See theorem `soundness` in appendix B for the full proof in Lean.

**Theorem 2.1.20 (Completeness)** The axiom system $\mathsf{K}$ over the language $\mathcal{L}$ is complete with respect to the class of all frames. That is, $\models \varphi$ implies $\vdash_{\mathsf{K}} \varphi$.

**Proof 2.1.20.1** See theorem `completeness` in appendix B for the full proof in Lean.

In general, axiom systems may be sound or complete with respect to a class of frames. For instance, $\mathsf{T}$ is sound and complete with respect to the class of reflexive frames, $\mathsf{S4}$ is sound and complete with respect to the class of reflexive and transitive frames, and $\mathsf{S5}$ is sound and complete with respect to the class whose relation is an equivalence relation. Given an arbitrary axiom system $\mathsf{AX}$ and a class of frames $\mathcal{F}$, we say that $\mathsf{AX}$ is sound and complete with respect to $\mathcal{F}$ if

$$\vdash_{\mathsf{AX}} \varphi \iff \mathcal{F} \models \varphi.$$

---

With the basics of modal logic having been outlined, one final question that may arise for the mature reader is the question of where modal logic fits in with respect to propositional logic and first-order logic.

The bounded universal quantification intrinsic to the clause for "box" in the definition of validity (2.1.4) means that modal logic can be interpreted in first order logic by interpreting the boxes with quantifiers over worlds. However, part of the reason modal logic is so interesting is because of its expressive power, for the converse does not hold: we cannot interpret first-order logic in modal logic. In sum, modal logic is closer to first-order logic than propositional logic in terms of expressivity, yet it preserves the computational complexity of propositional logic in terms of decidability.

Although I will not discuss the expressivity or decidability of modal logic in detail here, the interplay between these two features in modal logic can be understood through two-variable first-order logic and its relation to modal logic. The reader is encouraged to reference [1] and [18] for more on this topic.

## 2.2 Dynamic Epistemic Logic

We now turn our attention to one of the most active sub-fields of research within modal logic, dynamic epistemic logic. Here, the modal operator "box" takes on an epistemic interpretation – usually denoted $K_a\varphi$ – meaning, "agent $a$ knows that $\varphi$". Robert Stalnaker aptly presents this epistemic interpretation in his paper, *On Logics of Knowledge and Belief* [15] when he writes,

> Just as necessity is truth in all possible worlds, so knowledge is truth in all epistemically possible worlds. The assumption is that to have knowledge is to have a capacity to locate the actual world in logical space, to exclude certain possibilities from the candidates for actuality. The epistemic possibilities are those that remain after the exclusion, those that the knower cannot distinguish from actuality.

In addition to the epistemic interpretation of "box", action modalities of the form $[\varphi]\psi$ are added to the language as a way to capture communication events, where in this case $[\varphi]\psi$ means "after $\varphi$ is communicated, $\psi$ holds." This construction provides a logical framework for reasoning about how knowledge changes for systems of agents as communication events occur.

However, before I introduce the full logical framework, it is important to clarify what exactly is meant by the terms "knowledge", "agents", "change", and "communication events."

In this context, *knowledge* signifies any related body of data that can be held by an agent or subject. This is admittedly a loose characterization of knowledge, philosophically speaking. Nonetheless, it will prove fruitful here in that it will allow us to apply the theory of dynamic epistemic logic to a wide variety of agents capable of such unrefined and diverse forms of knowledge. As such, knowledge could be thought of as something more like *information*, and the two terms may be used interchangeably in the discussion that follows. What is important to grasp about this characterization of knowledge (or information) is that it is a coherent body of data, rather than a jumble of miscellaneous facts.

Because knowledge is taken to have such wide scope, the term *agents* may on one end of the spectrum refer to human beings replete with complex mental activity. On the other end of the spectrum agents may represent something more abstract, such as vertices in a social network or servers in a distributed system. Agents could also represent self-driving cars that collect data from their environment and make decisions about how to best interact with that environment and other vehicles on the road.[12] What unites all of these instances of agenthood is the fact that information is relative to an agent, and different agents possess different information to be shared with other agents.

Finally, the *change* of information that is of interest within dynamic epistemic logic – and that which adds a dynamic aspect to the epistemic framework – occurs via *communication events* between agents. An important aspect of these communication events is that they usually do not change the facts under consideration, but rather the knowledge that agents hold about these facts. This results in what is called epistemic updates for agents, or formulas that become true or false after communication events.[13] The theory that concerns knowledge change for agents due to a change in underlying facts is beyond the scope of this thesis, as well as the theory of dynamic epistemic logic more broadly.

### 2.2.1 Private and Public Announcements

Numerous forms of communication are studied withing the dynamic epistemic framework, ranging from public communications to private. In the literature such communications are commonly called *announcements*.

For example, a private announcement $\varphi$ is one that is made to a single agent $a$. No other agents within the group of agents know the contents of $\varphi$, and more importantly, other agents may not even consider it possible that $a$ knows $\varphi$ either.

[12] Or interconnected subsystems of a larger system, as was seen in the example in the introduction.

[13] One example of an update is the *unsuccessful update*. If someone publicly announces to a group, $p \wedge \neg K_a p$, i.e.,"$p$ is true, but Alice does not know it", then this announcement immediately becomes false if Alice is a member of the group. For indeed, through the announcement itself, Alice comes to know that $p$ is true. This is in contrast to the *successful update* in which some formula $\varphi$ remains true after it is announced.

In real world applications, private communications can occur in fields such as computer security and encryption. For instance, one might want to ensure that a key cannot be decrypted by an intruder. This is equivalent in the language of dynamic epistemic logic to asking what an outside agent can know based on the private communication between the agents with the keys, as well as existing facts about the world, such as the strengths and weaknesses of the underlying security system.

On the other hand, public announcements are truthful communications made to all members in a group. This fragment of dynamic epistemic logic (called *public announcement logic*) is useful for one because it provides a way to reason about problems concerning the common knowledge and distributed knowledge of a group.

For example, common knowledge of $\varphi$ means that every agent in a given group of agents $G$ knows that $\varphi$, and everybody knows that everybody knows $\varphi$, and so on, ad infinitum. Alternatively, distributed knowledge of $\varphi$ means that, if all agents in a group $G$ were to pool their knowledge, then together they would collectively know $\varphi$. To formally reason about common or distributed knowledge, new symbols $C_G\varphi$ and $D_G\varphi$ are added to the language of public announcement logic, along with semantic interpretations of their meaning and syntactic rules for their use.

However, even if common or distributed knowledge are not taken into account, many interesting questions can be posed and answered via public announcement logic.[14] Moreover, public announcement logic without common knowledge is interesting for theoretical reasons relating to expressivity and completeness, which I shall expound upon shortly. Thus, my primary focus in this part of the thesis centers around formalizing the version of public announcement logic without common knowledge. To the syntax and semantics of the language we now turn.

### 2.2.2 Syntax and Semantics

The language of public announcement logic without common knowledge (henceforth referred to simply as public announcement logic or PAL) rests on the language of epistemic logic. In order to develop the theory of PAL, I shall draw parallels between the two systems in the section that follows by stating all of the essential definitions of the two systems in tandem.

To begin, the language of epistemic logic is essentially the same as the language of basic modal logic, with the exception that the "box" operator is denoted $K$, and there are multiple instances of these operators, one for each agent $a$.

**Definition 2.2.1 (The Language of Epistemic Logic)** The well-formed formulas $\varphi$ of epistemic logic are given recursively by the following rules:

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \psi \mid K_a\varphi$$

where $a \in A$ ranges over a finite set of agents and $p \in \text{PROP}$ ranges over a countable set of primitive propositions. In the discussion that follows, this language is denoted $\mathcal{L}_K$.

[14] For example, PAL's most common motivational examples include the Muddy Children Puzzle, the Sum and Product Puzzle, and Cheryl's Birthday Puzzle.[4]

To define the language of PAL, we include a new symbol, [ ], used to denote communication events. This language is specified as follows.

**Definition 2.2.2 (The Language of Public Announcement Logic)** The well-formed formulas $\varphi$ of public announcement logic are given recursively by the following rules:

$$\varphi := p \mid \neg\varphi \mid \varphi \wedge \psi \mid K_a\varphi \mid [\varphi]\psi$$

where $a \in A$ ranges over a finite set of agents and $p \in \text{PROP}$ ranges over a countable set of primitive propositions. In the discussion that follows, this language is denoted $\mathcal{L}_{K[]}$.

Since the knowledge operator is indexed by agents in both languages, the binary relation $R^A$ must be as well. That is, instead of a single binary relation over all agents, there exists a binary relation $R^A(a) \subset W \times W$ for every $a \in A$, as seen in the following definitions of Kripke frames and Kripke models[15]

**Definition 2.2.3 (Kripke Frame)** Given a finite set of agents $A$, a Kripke frame is a pair $F = (W, R^A)$ where $W$ denotes a non-empty set of possible worlds and $R^A \subseteq W \times W$ denotes a function that yields for every $a \in A$ an accessibility relation $R^A(a) \subset W \times W$. Note that we will often write $R_a$ rather than $R^A(a)$, and $xR_ay$ rather than $xR^A(a)y$.

**Definition 2.2.4 (Kripke Model)** Given a finite set of agents $A$ and a countable set of primitive propositions PROP, a Kripke model $M = ((W, R^A), v)$ is a tuple where $F = (W, R^A)$ denotes a frame and $v : \text{PROP} \to 2^W$ denotes a valuation function over the primitive propositions.

Next, validity for epistemic logic essentially follows from validity for basic modal logic, the only exception being that it is now indexed by agents.

**Definition 2.2.5 (Validity for Epistemic Logic)** Let $x$ be a world in a model $M = ((W, R^A), v)$. We say that formula $\varphi$ is valid for agent $a$ at world $x$ in $M$ whenever

$$\begin{array}{lll}
(M,x) \models p & \text{iff} & x \in v(p) \\
(M,x) \models \neg\varphi & \text{iff} & (M,x) \not\models \varphi \\
(M,x) \models \varphi \wedge \psi & \text{iff} & (M,x) \models \varphi \text{ and } (M,x) \models \psi \\
(M,x) \models K_a\varphi & \text{iff} & \forall y \in W,\ xR_ay \text{ implies } (M,y) \models \varphi
\end{array}$$

On the other hand, validity for public announcement logic is more complicated. Public announcement logic is an example of a *relation-changing logic*. That is, it is a logic with the ability to update a model while evaluating the truth of a formula [9]. Relation-changing logics are interesting because they allow us to assess how a relational structure evolves through the application of various operations – in this case, the announcement formula $[\varphi]\psi$. To more formally describe how models evolve via announcements, we have the following definition.

[15]Definitions 2.2.3 and 5.1.2 apply identically to both the languages of epistemic logic and public announcement logic.

**Definition 2.2.6 (Validity for Public Announcement Logic)** Let $x$ be a world in a model $M = ((W, R^A), v)$. We say that formula $\varphi$ is valid for agent $a$ at world $x$ in $M$ whenever

$$\begin{array}{lll} (M,x) \models p & \text{iff} & x \in v(p) \\ (M,x) \models \neg\varphi & \text{iff} & (M,x) \not\models \varphi \\ (M,x) \models \varphi \wedge \psi & \text{iff} & (M,x) \models \varphi \text{ and } (M,x) \models \psi \\ (M,x) \models K_a\varphi & \text{iff} & \forall y \in W,\ xR_ay \text{ implies } (M,y) \models \varphi \\ (M,x) \models [\varphi]\psi & \text{iff} & (M,x) \models \varphi \text{ implies } (M|\varphi, x) \models \psi \end{array}$$

where $M|\varphi = (W', R'_a, v')$ is such that

$$\begin{aligned} W' &= [\![\varphi]\!]_M \\ R'_a &= R_a \cap ([\![\varphi]\!]_M \times [\![\varphi]\!]_M) \\ v'(p) &= v(p) \cap [\![\varphi]\!]_M \end{aligned}$$

In essence, the public announcement of $[\varphi]$ is a state transformer; it restricts the set of states to only those states where $\varphi$ holds. The dual of $[\varphi]$ is denoted $\langle\varphi\rangle$, and its semantics are given by:

$$(M,x) \models \langle\varphi\rangle\psi \quad \text{iff} \quad (M,x) \models \varphi \text{ and } (M|\varphi, x) \models \psi.$$

As mentioned previously, public announcement logic is interesting for theoretical reasons relating to expressivity and completeness. These ideas will be further expounded in the following section.

### 2.2.3 Soundness and Completeness

The axiom system PA for public announcement logic rests on the axiom system S5 for epistemic logic indexed over agents, which is given by the following definition.

**Definition 2.2.7 (System S5)** Given a set of agents $A$ and primitive propositions $P$, the axiomatic system S5 is comprised of:

- all instances of propositional tautologies,
- all instances of the schemes:

  $K_a(\varphi \to \psi) \to (K_a\varphi \to K_a\psi)$, (DISTRIBUTION OF $K_a$ OVER $\to$)

  $K_a\varphi \to \varphi$, (TRUTH)

  $K_a\varphi \to K_aK_a\varphi$, (POSITIVE INTROSPECTION)

  $\neg K_a\varphi \to K_a\neg K_a\varphi$, (NEGATIVE INTROSPECTION)
- from $\varphi$ and $\varphi \to \psi$, we may infer $\psi$, (MODUS PONENS)
- from $\varphi$, we may infer $K_a\varphi$. (NECESSITATION)

System S5 contains several axioms schemas regarding knowledge that should hopefully accord with our intuitions. For example, TRUTH says that if an agent knows $\varphi$, then $\varphi$

is true; and POSITIVE INTROSPECTION says that if an agents knows $\varphi$, then they *know* that they know $\varphi$. The axiom for NEGATIVE INTROSPECTION is much more debated within the literature; some formal epistemologists argue that it represents too idealized a form of knowledge, as pointed out by Stalnaker [15]. However, the authors of the textbook *Dynamic Epistemic Logic* [16] chose to take this axiom, so my formalization follows suit.

It is also worth noting the necessitation rule here. Necessitation may not seem entirely plausible when interpreted over knowledge and agents. Indeed, as Stalnaker [15] again points out, no matter how epistemically possible worlds are selected, all of the logical truths will be true in them. By necessitation then, all knowers would hence know all logical truths, which seems absurd. However, others argue that it is indeed a plausible rule to have in a proof system, as it formally echoes the idea that if something is demonstrable by logical reflection alone then one can necessarily know it is true.

In addition to these axioms, the system PA contains additional axioms for proving formulae involving announcements. These axioms govern how announcement interacts with the other connectives. For example, the last axiom (ANNOUNCEMENT COMPOSITION) is important because it reduces a composition of two announcements into a conjunction of announcements, so it is helpful when deriving theorems involving multiple announcements.

**Definition 2.2.8 (System PA)** Given a set of agents $A$ and primitive propositions $P$, the axiomatic system PA is comprised of:

- all instances of propositional tautologies,
- all instances of the schemes:

| | |
|---|---|
| $K_a(\varphi \to \psi) \to (K_a\varphi \to K_a\psi)$, | (DISTRIBUTION OF $K_a$ OVER $\to$) |
| $K_a\varphi \to \varphi$, | (TRUTH) |
| $K_a\varphi \to K_aK_a\varphi$, | (POSITIVE INTROSPECTION) |
| $\neg K_a\varphi \to K_a\neg K_a\varphi$, | (NEGATIVE INTROSPECTION) |
| $[\varphi]p \leftrightarrow (\varphi \to p)$, | (ATOMIC PERMANENCE) |
| $[\varphi]\neg\psi \leftrightarrow (\varphi \to \neg[\varphi]\psi)$, | (ANNOUNCEMENT AND NEGATION) |
| $[\varphi](\psi \wedge \chi) \leftrightarrow ([\varphi]\psi \wedge [\varphi]\chi)$, | (ANNOUNCEMENT AND CONJUNCTION) |
| $[\varphi]K_a\psi \leftrightarrow (\varphi \to K_a[\varphi]\psi)$, | (ANNOUNCEMENT AND KNOWLEDGE) |
| $[\varphi][\psi]\chi \leftrightarrow [\varphi \wedge [\varphi]\psi]\chi$, | (ANNOUNCEMENT COMPOSITION) |

- from $\varphi$ and $\varphi \to \psi$, we may infer $\psi$, (MODUS PONENS)
- from $\varphi$, we may infer $K_a\varphi$. (NECESSITATION)

Recall from section 2.1.3, system S5 over the basic modal language $\mathcal{L}$ is sound and complete with respect to the class of frames satisfying an equivalence relation, $\mathcal{F}_{eq}$ (see theorem `S5_soundness` in section 3.2 and theorem `S5_completeness` in appendix B for the full proof in Lean). Likewise, the same axiomatic system S5 in the epistemic language $\mathcal{L}_K$ indexed over agents is sound and complete with respect to the class $\mathcal{F}_{eq}$ (see proofs `soundnessS5` and `completenessS5` in appendix C for the full proof in Lean). What may be surprising, however, is that the axiomatic system PA over the public announcement language $\mathcal{L}_{K[]}$ is also sound and complete with respect to the *exact same* class $\mathcal{F}_{eq}$.

**Theorem 2.2.9 (Completeness)** The axiom system $\mathsf{PA}$ over the language $\mathcal{L}_{K[]}$ is sound and complete with respect to the class of frames $\mathcal{F}_{eq}$ whose relation is an equivalence relation. That is, $\vdash_{\mathsf{PA}} \varphi$ iff $\mathcal{F}_{eq} \models \varphi$.

**Proof 2.2.9.1** See theorem `soundnessPA` in appendix C and `completenessPA` in section 3.3 for the full proofs in Lean.

Why is it that these two axiom systems, one an extension of the other, are sound and complete with respect to the same class of frames? The reason is because the languages $\mathcal{L}_K$ and $\mathcal{L}_{K[]}$ have the same expressive power. That is, for any formula in $\mathcal{L}_{K[]}$, we can find a formula in $\mathcal{L}_K$ that is equivalent to it (the converse is trivial). Thus, to prove completeness for system $\mathsf{PA}$ with respect to the class of frames $\mathcal{F}_{eq}$, it suffices to define a translation function $t : \mathcal{L}_{K[]} \to \mathcal{L}_K$ and show that every formula is provably equivalent to its translation. The result then follows from completeness of $\mathsf{S5}$ with respect to $\mathcal{F}_{eq}$ over the epistemic language $\mathcal{L}_K$.

This translation function is defined as follows.

**Definition 2.2.10 (Translation)** The translation $t : \mathcal{L}_{K[]} \to \mathcal{L}_K$ is defined recursively as:

$$\begin{aligned}
t(p) &= p \\
t(\neg\varphi) &= \neg t(\varphi) \\
t(\varphi \wedge \psi) &= t(\varphi) \wedge t(\psi) \\
t(K_a\varphi) &= K_a t(\varphi) \\
t([\varphi]p) &= t(\varphi \to p) \\
t([\varphi]\neg\psi) &= t(\varphi \to \neg[\varphi]\psi) \\
t([\varphi](\psi \wedge \chi)) &= t([\varphi]\psi \wedge [\varphi]\chi) \\
t([\varphi]K_a\psi) &= t(\varphi \to K_a[\varphi]\psi) \\
t([\varphi][\psi]\chi) &= t([\varphi \wedge [\varphi]\psi]\chi)
\end{aligned}$$

As stated above, we intend to show that every formula is provably equivalent to its translation. The proof proceeds by induction, but not by induction on formulas as one might assume. This will not suffice for the case at hand, for during the inductive step of a proof by induction on formulas, one would need the inductive hypothesis for subformulas of that formula. However, $([\varphi]\psi \wedge [\varphi]\chi)$ is not a subformula of $[\varphi](\psi \wedge \chi)$, for example. So instead, we must define a complexity measure and proceed by proof by induction on complexity of formulas.

**Definition 2.2.11 (Complexity)** The complexity $c : \mathcal{L}_{K[]} \to \mathcal{N}$ is defined recursively as:

$$\begin{aligned}
c(p) &= 1 \\
c(\neg\varphi) &= 1 + c(\varphi) \\
c(\varphi \wedge \psi) &= 1 + \max(c(\varphi), c(\psi)) \\
c(K_a\varphi) &= 1 + c(\varphi) \\
c([\varphi]\psi) &= (4 + c(\varphi)) \cdot c(\psi)
\end{aligned}$$

With the complexity measure in hand, we can show that each formula is provable equivalent to its translation.

**Lemma 2.2.12** For all formulas $\varphi \in \mathcal{L}_{K[]}$ it is the case that

$$\vdash \varphi \leftrightarrow t(\varphi).$$

**Proof 2.2.12.1** By induction on complexity $c(\varphi)$. See proofs `equiv_translation_aux` and `equiv_translation` in appendix C.

This completes chapter 2. In the next chapter, I will describe the formalization in Lean.

# Chapter 3

# Formalization

This chapter presents the formalization in the Lean theorem prover of the modal theory explored in chapter 2. In the following sections, Lean code will be used to illustrate design choices as well as to give a feel for what mathematical definitions and proofs look like in Lean compared to those that are typeset or written by hand.

This formalization assumes a variety of general mathematical results that depend on some data structures not present in the standard Lean library. For this reason, I shall import the library of formalized mathematics for Lean called *mathlib* [13].

This project contains about 5000 lines of code, and all proofs have been typechecked with Lean 3.18.4. The reader is encouraged to consult the source code that is described in this thesis, which is publicly available online at: `https://www.github.com/paulaneeley/modal`.

To begin, section 3.1 provides a high-level overview of the Lean theorem prover. Next, section 3.2 details the definitions and proofs pertaining to modal definability and undefinability, as well as Henkin-style completeness proofs for the propositional modal logics $\mathsf{K}$, $\mathsf{T}$, $\mathsf{S4}$, and $\mathsf{S5}$.[1] Finally, section 3.3 details the completeness proof by translation for public announcement logic.

## 3.1 Lean

Lean is an open source proof assistant first developed in 2013 by Leonardo de Moura at Microsoft Research. Lean combines interactive theorem proving with automated reasoning capabilities in a dependently typed programming language called the Calculus of Inductive Constructions. Since its inception, the list of developers and contributors to Lean has grown to produce a thriving online community of members.

At its core, Lean consists of a general specification language and axiomatic framework. As such, users have the flexibility to embed domain-specific logics within the host language of Lean as collections of definitions and formulas. Embeddings may be deep or shallow: shallow embeddings provide syntactic translations ideal for proving results in a logic's object language. Conceptually, this type of embedding describes the semantics of a given domain in a data type in order to provide a fixed interpretation for it. A deep embedding, on the

[1] A proof of completeness for a given logic is called a *Henkin-style* proof whenever it relies on nonconstructive methods to build models out of maximal consistent sets of formulas [5].

other hand, enable proofs of metalogical results (such as cut-elimination or completeness) via proof by structural induction on abstract syntax trees. Conceptually, this type of embedding describes the semantics of the *operations* on the domain, enabling variable interpretations. Embedding a logic within a specification language such as Lean can serve many purposes, such as formally reasoning about the correctness of applications of that logic through the use of interactive theorem proving.

Lean has become popular among working mathematicians as an interactive theorem prover in part for its ability to support classical as well as constructive reasoning. Classical reasoning and non-computable constructions are ubiquitous in this formalization, such as in the use of Zorn's lemma in the proof of Lindenbaum's lemma for completeness.

Theorem proving in Lean adheres to two styles. The first involves the use of *proof terms*, which are representations of mathematical proofs that can be constructed directly in the functional language [2]. Alternatively, proofs can be constructed using *tactics*, which are imperative commands that instruct or tell Lean how to build a proof. Lean is also a flexible environment in that proof terms and tactics can be mixed together in the same proof. Many of the proofs in this formalization are tactic-style proofs, but this is merely a matter of preference for me as a Lean user.

## 3.2 Modal Logic

### 3.2.1 Syntax and Semantics

To define the language of modal logic in Lean, I have chosen to work with the atomic formula falsum ($\bot$) and a countable set of propositional variables indexed by the natural numbers ($p_n$). In addition, I have chosen to work with a non-minimal set of connectives that includes conjunction (&) implication ($\supset$), and necessitation ($\Box$). While I am aware that a minimal set of connectives would have sufficed (for example, either & or $\supset$, but not both), and that the use of a non-minimal set requires one to prove an additional clause in proofs that rely on induction on formulas, certain very complex proofs were actually made simpler by including this extra connective, so this trade-off was made. Hence, the language of basic modal logic is given by the following inductive definition in Lean.

```
inductive form : Type
  | bot                 : form
  | var  (n : nat)      : form
  | and  (φ ψ : form)   : form
  | impl (φ ψ : form)   : form
  | box  (φ : form)     : form
```

Using one of the five constructors above (`bot`, `var`, `and`, `impl`, and `box`) is the only way to construct a term of type `form`. Other logical connectives such as $\vee$ and $\neg$ are defined as abbreviations in the usual way based on these core connectives, where

```
notation `¬` φ   := form.impl φ form.bot
notation φ `∨` ψ := ((¬φ) ⊃ ψ)
```

Because the theorems we want to prove are metatheoretical, I chose to encode the syntax using a deep embedding rather than a shallow one. This allows us to prove metatheorems via structural induction on abstract syntax trees.

In terms of the basic semantics, Kripke frames are formalized in Lean as `structures`. These are non-recursive inductive types with only one constructor. In this instance, a `frame` is a 3-tuple consisting of states of some type, a hypothesis that the states are non-empty, and a binary relation between states.[2]

```
structure frame :=
  (states : Type)
  (h : inhabited states)
  (rel : states → states → Prop)
```

Since many of the model-theoretic results in this formalization centered around *frame* definability and undefinability, I chose not to formally define Kripke *models* in Lean. Instead, when one desires to work with models rather than frames in this formalization, one can merely define a valuation function `v : nat → f.states → Prop` that corresponds to whatever term `f : frame` is in the current context.

This concept is nicely illustrated by the following definition. The notion of validity of a formula at a world in a model is recursively defined by cases in Lean as a function which takes a frame `f` and a valuation `v` as inputs and returns a function `f.states → form → Prop`.

```
def forces (f : frame) (v : nat → f.states → Prop) :
f.states → form → Prop
  | x (bot)      := false
  | x (var n)    := v n x
  | x (and φ ψ)  := (forces x φ) ∧ (forces x ψ)
  | x (impl φ ψ) := (forces x φ) → (forces x ψ)
  | x (box φ)    := ∀ y, f.rel x y → forces y φ
```

Next, we can define the global semantic consequence from a set of formulas (or context) Γ. A context is defined in Lean using sets, which in this case are functions of type `form → Prop` as follows.

```
@[reducible] def ctx : Type := set form
notation Γ `∪` φ := set.insert φ Γ
```

With contexts defined, we can say what it means for a formula $\varphi$ to be a global semantic consequence of a set of formulas Γ in Lean with the following two definitions. The first, `forces_context` says that every formula $\varphi$ in Γ is true at every world `x` in a frame `f`. The definition `global_sem_csq` then says that if every formula in Γ is true for some frame `f` and valuation `v`, then the truth of $\varphi$ follows at every world `x` in `f`.

```
def forces_ctx (f : frame) (v : nat → f.states → Prop) (Γ : ctx) :=
  ∀ φ, ∀ x, φ ∈ Γ → forces f v x φ
```

[2]For instance, when one instantiates a term of type `frame`, one must provide a type for the states, a proof that the type is non-empty (i.e., inhabited), and a relation between the states.

```
def global_sem_csq (Γ : ctx) (F : set frame) (φ : form) :=
  ∀ f ∈ F, ∀ v, forces_ctx f v Γ → ∀ x, forces f v x φ
```

### 3.2.2 Frame Definability and Undefinability

To extend the definition of local validity given above to more global contexts, in Lean we can say that a formula $\varphi$ is valid:

- In a model $M$, if

```
def m_valid (φ : form) (f : frame) (v : nat → f.states → Prop) :=
  ∀ x, forces f v x φ
```

- In a frame $F$, if

```
def f_valid (φ : form) (f : frame) :=
  ∀ v x, forces f v x φ
```

- On a class of frames $\mathcal{F}$, if

```
def F_valid (φ : form) (F : set frame) :=
  ∀ f ∈ F, ∀ v x, forces f v x φ
```

- Universally, if

```
def u_valid (φ : form) :=
  ∀ f v x, forces f v x φ
```

The definitions of definability and undefinability are written in Lean almost as simply and straightforwardly as they are by hand (see the informal definitions in section 2.1.2). Indeed, for definability and undefinability, we have,

```
def defines (φ : form) (F : set frame) :=
  ∀ f, f ∈ F ↔ f_valid φ f

def undefinable (F : set frame) :=
  ∀ φ, ¬ defines φ F
```

We can also define the theory at a world in a model via the following definition.

```
def theory_at (f : frame) (v : nat → f.states → Prop)
(x : f.states) : set form := { φ : form | forces f v x φ}
```

Although these definitions are written in Lean quite succinctly, the definitions and proofs of invariance under disjoint union, generated submodels, bisimulation, and surjective bounded morphisms are somewhat lengthy. The reader is encouraged to consult appendix A or the source code on Github for the full definitions and proofs.

### 3.2.3 Soundness and Completeness

We now have a rich semantics for modal logic formalized in Lean. In order to prove soundness and completeness results for various proof systems with respect to their corresponding classes of frames, I need to provide definitions of these proof systems in Lean, as well as to explain the notion of provability in Lean. In addition, I must define what it means for a formula $\varphi$ to be provable from a set of formulas $\Gamma$.[3]

I will begin with the basic proof system K, which is defined inductively in Lean as follows. Here is how to read the first case, `ax`: given a context `AX`, a formula $\varphi$, and the hypothesis that $\varphi$ is in `AX`, then one can construct a term of type `prfK AX` $\varphi$, which represents an object-language proof of $\varphi$. Likewise, the second case, `pl1`, says that given a context `AX` and formulas $\varphi$ and $\psi$, then one can construct a term of type `prfK AX` $\varphi \supset (\psi \supset \varphi)$. The other axiomatic cases follow similarly. The rule of inference, `mp`, is read as follows: given a context `AX`, formulas $\varphi$ and $\psi$, and the hypotheses `hpq : prfK AX` $(\varphi \supset \psi)$ and `hp : prfK AX` $\varphi$, then one can construct a term of type `prfK AX` $\psi$. The other rule of inference, `nec` follows similarly.

```
inductive prfK : ctx → form → Prop
  | ax {AX} {φ} (h : φ ∈ AX) : prfK AX φ
  | pl1 {AX} {φ ψ}           : prfK AX φ ⊃ (ψ ⊃ φ)
  | pl2 {AX} {φ ψ χ}         : prfK AX (φ ⊃ (ψ ⊃ χ)) ⊃ ((φ ⊃ ψ) ⊃ (φ ⊃ χ))
  | pl3 {AX} {φ ψ}           : prfK AX (¬φ ⊃ ¬ψ) ⊃ ((¬φ ⊃ ψ) ⊃ φ)
  | pl4 {AX} {φ ψ}           : prfK AX φ ⊃ (ψ ⊃ (φ & ψ))
  | pl5 {AX} {φ ψ}           : prfK AX (φ & ψ) ⊃ φ
  | pl6 {AX} {φ ψ}           : prfK AX (φ & ψ) ⊃ ψ
  | pl7 {AX} {φ ψ}           : prfK AX (¬φ ⊃ ¬ψ) ⊃ (ψ ⊃ φ)
  | kdist {AX} {φ ψ}         : prfK AX □(φ ⊃ ψ) ⊃ ((□φ) ⊃ (□ψ))
  | mp {AX} {φ ψ}
    (hpq: prfK AX (φ ⊃ ψ))
    (hp : prfK AX φ)         : prfK AX ψ
  | nec {AX} {φ}
    (hp: prfK AX φ)          : prfK AX □φ
```

Remember that back in definition 2.1.17 we stated that the proof system $K$ contained all instances of propositional tautologies. In Lean however, there is no way to define all instances of propositional tautologies within the proof system `prfK`. Instead, I have chosen a small set of axioms, `pl1`-`pl7`, from which all propositional tautologies of classical logic follow. For a complete listing of all the propositional tautologies that I proved in the object language, see the file `syntaxlemmas.lean`.

To build the proof systems T, S4, and S5 from K, we have the following simple definitions in Lean.

[3] A context $\Gamma$ could constitute hypotheses from which we aim to prove a given formula $\varphi$. On the other hand, a context could represent additional axioms from which we can expand the system K to the systems T, S4, or S5.

```
def T_axioms  : ctx := {φ : form | ∃ ψ, φ = (□ψ ⊃ ψ)}
def S4_axioms : ctx := T_axioms ∪ {φ : form | ∃ ψ, φ = (□ψ ⊃ □(□ψ))}
def S5_axioms : ctx := T_axioms ∪ {φ : form | ∃ ψ, φ = (◇ψ ⊃ □(◇ψ))}
```

Since the proof systems are defined inductively in Lean, there is no need to formally define a notion of deduction in Lean as we did in the informal theory (see definition 2.1.18 in section 2.1). Instead, provability of the axiom systems in Lean is informally defined inductively as follows: given a proof system AX (either K, T, S4, or S5), a formula $\varphi$ is provable from AX if either:

- $\varphi$ is an axiom,
- $\varphi$ follows from provable formulas $\psi$ and $\chi$ by modus ponens,
- $\varphi$ follows from provable formula $\psi$ by necessitation.

For example, to show that the axiom system K is sound with respect to the class of all frames, the proof in Lean proceeds by induction on the provability relation. For the full proof, see appendix B.

```
theorem soundness (AX : ctx) (F : set (frame)) (φ : form) :
  prfK AX φ → global_sem_csq AX F φ := ...
```

Notice that the proof above was carried out with respect to an arbitrary axiom system `AX` and class of frames `F`. As such, it technically not only specifies soundness for system K with respect to the class of all frames, it specifies soundness for a whole family of proof systems parameterized by `AX` and `F`. Thus, soundness for proof systems T, S4, and S5 follows (with the help of a few additional lemmas and definitions) by providing different proofs systems as the argument to `AX`, and different classes of frames as the argument to `F`. The statements of these theorems are written as follows.

```
theorem T_soundness (φ : form) :
  prfK T_axioms φ → global_sem_csq T_axioms ref_class φ := ...

theorem S4_soundness (φ : form) :
  prfK S4_axioms φ → global_sem_csq S4_axioms ref_trans_class φ := ...

theorem S5_soundness (φ : form) :
  prfK S5_axioms φ → global_sem_csq S5_axioms equiv_class φ := ...
```

The reader is encouraged to see appendix B or the full formalization on Github for the detailed proofs.

---

With soundness results formalized in Lean, we now turn our attention to the proof of completeness. This proof uses contraposition, and is therefore non-constructive. We shall make use of a canonical model construction, and since canonical models rely on maximally consistent sets, we begin our account with some definitions.

Consider a finite context $\Gamma = \{\varphi_1, \varphi_2, ..., \varphi_n\}$. We say that $\Gamma$ is AX-consistent if $\nvdash_{\mathsf{AX}} \neg(\varphi_1 \wedge \varphi_2 \wedge ... \wedge \varphi_n)$. This is represented in Lean with the following two definitions.

```
def fin_conj : list form → form
  | []       := ⊤
  | (φ::φs) := φ & (fin_conj φs)

def fin_ax_consist (AX : ctx) (L : list form) :=
  ¬ prfK AX (fin_conj L ⊃ ⊥)
```

The first definition, `fin_conj`, represents a finite conjunction of formulas using lists in Lean. It is defined recursively: a formula $\varphi$ cons-ed with a list of formulas $\varphi$s evaluates to the conjunction of $\varphi$ and (`fin_conj` $\varphi$s). The base case is, of course, necessary, but also purposely vacuous; for any formula $\varphi$, the truth value of $(\varphi\&\top)$ semantically evaluates to whatever the truth value of $\varphi$ is. The definition `fin_ax_consist` then says that it is not the case that the finite conjunction of formulas `L` proves $\bot$.

An *arbitrary* context $\Gamma$ is called AX-consistent if every finite subset of $\Gamma$ is AX-consistent. For this, we can write the following definition in Lean.

```
def ax_consist (AX Γ : ctx) :=
  ∀ L : list form, (∀ ψ ∈ L, ψ ∈ Γ) → fin_ax_consist AX L
```

Finally, we can define a context $\Gamma$ to be maximally AX-consistent if it is AX-consistent and every proper superset of $\Gamma$ is not AX-consistent.

```
def max_ax_consist (AX Γ : ctx) :=
  ax_consist AX Γ ∧ (∀ Γ', Γ ⊂ Γ' → ¬ ax_consist AX Γ')
```

Since each world in the canonical model construction corresponds to a maximally consistent set of formulas, an important step in the proof of completeness is to prove Lindenbaum's Lemma, which states that any consistent set of formulas can be extended to a maximally consistent set. For this, we have the following statement of the theorem in Lean.[4] Due to its length, the reader is encouraged to consult the full proof in appendix B.

```
lemma lindenbaum (AX Γ : ctx) (hax : ax_consist AX Γ) :
  ∃ Γ', max_ax_consist AX Γ' ∧ Γ ⊆ Γ' := ...
```

Lindenbaum's lemma gives us the following important corollary, which states that a maximally AX-consistent set indeed exists.

```
lemma max_ax_exists (AX : ctx) (hax : sem_cons AX) :
  ∃ Γ : ctx, max_ax_consist AX Γ := ...
```

With these definitions in hand, we can build the canonical model. As with simpler semantic definitions in Lean outlined above, we shall again define a canonical *frame* and a canonical *valuation* separately.

[4]This construction departed from the standard construction often seen in introductory logic textbooks using languages of countable cardinalities. Instead, I chose here to use Zorn's lemma to generalize this construction to languages of arbitrary cardinalities.

```
def canonical (AX : ctx) [hax : sem_cons AX] : frame :=
{
  states := {xΓ : ctx // max_ax_consist AX xΓ},
  h :=
  begin
    have h1 := max_ax_exists AX hax,
    choose Γ h1 using h1,
    exact ⟨⟨Γ, h1⟩⟩
  end,
  rel := λ xΓ yΔ, ∀ φ : form, □φ ∈ xΓ.val → φ ∈ yΔ.val
}
```

In other words, states in the canonical model are contexts that are maximally AX-consistent. The hypothesis that the set of worlds is non-empty is satisfied by the corollary of Lindenbaum's lemma above, which states that a maximally AX-consistent set exists. The accessibility relation states that whenever the world $x\Gamma$ has in its context the formula $\Box\varphi$, then the only accessible worlds from $x\Gamma$ are worlds $y\Delta$ that have in their contexts the formula $\varphi$. Finally, the implicit argument `[hax : sem_cons AX]` marked with square brackets informs the Lean elaborator that the hypothesis inside should be inferred by the type class mechanism. The hypothesis, `sem_cons AX`, states that $\bot$ is not a global semantic consequence of the axiom system AX.

The canonical valuation is defined in Lean as follows: `p n` is true at exactly those worlds having maximally AX-consistent contexts that contain `p n`.

```
def val_canonical (AX : ctx) [hax : sem_cons AX] :
nat → (canonical AX).states → Prop :=
  λ n, λ xΓ : (canonical AX).states, (p n) ∈ xΓ.val
```

These definitions are used in order to prove the Truth lemma, which states that for all formulas $\varphi$, we have that $\varphi$ is contained in the context of each world in the canonical model if and only if $\varphi$ is valid at that world in the canonical model. For the full proof, see appendix B.

```
lemma truth (AX : ctx) (hax : sem_cons AX) (xΓ : (canonical AX).states) :
  ∀ φ, forces (canonical AX) (val_canonical AX) xΓ φ ↔ (φ ∈ xΓ.val) := ...
```

The completeness theorem follows as a corollary to the Truth lemma. For the full proof, see appendix B.

```
theorem completeness (AX : ctx) (hax : sem_cons AX) (φ : form) :
  global_sem_csq AX all_class φ → prfK AX φ := ...
```

To extend completeness results for T, S4, and S5, we must show that the canonical model for T is reflexive, the canonical model for S4 is reflexive and transitive, and the canonical model for S5 is an equivalence relation. These canonical models are defined in Lean as follows.

```
def T_canonical  : frame := @canonical T_axioms  sem_consT

def S4_canonical : frame := @canonical S4_axioms sem_consS4

def S5_canonical : frame := @canonical S5_axioms sem_consS5
```

The statement of the proofs regarding their accessibility relations are written below. For the full proofs, as well as the proofs of completeness for these systems, see appendix B.

```
lemma T_reflexive : T_canonical ∈ ref_class := ...

lemma S4_reftrans : S4_canonical ∈ ref_trans_class :=

lemma S5_equiv    : S5_canonical ∈ equiv_class := ...
```

This concludes the description of the formalization of basic modal logic in Lean. In the next section, I will address the formalization of dynamic epistemic logic in Lean.

## 3.3 Dynamic Epistemic Logic

### 3.3.1 Syntax and Semantics

We now begin our account of the formalization of public announcement logic without common knowledge in Lean. To reiterate from section 2.2.1, the language of public announcement logic builds on the language of epistemic logic, so in order to describe the formalized theory of public announcement logic, I shall also state many of the formalized definitions for epistemic logic in parallel. I will begin by defining the two languages, `form` for epistemic logic and `formPA` for public announcement logic.

```
inductive form (agents : Type) : Type
  | bot                 : form
  | var  (n : nat)      : form
  | and  (φ ψ : form)   : form
  | impl (φ ψ : form)   : form
  | box  (a : agents)
         (φ : form)     : form

inductive formPA (agents : Type) : Type
  | bot                     : formPA
  | var    (n : nat)        : formPA
  | and    (φ ψ : formPA)   : formPA
  | impl   (φ ψ : formPA)   : formPA
  | box    (a : agents)
           (φ : formPA)     : formPA
  | update (φ ψ : formPA)   : formPA
```

For both languages, I have chosen to use the atomic formula falsum ($\bot$) and a countable set of propositional variables indexed by the natural numbers ($p_n$). In addition, I have chosen to work with a non-minimal set of connectives that includes conjunction (&) implication ($\supset$),and necessitation ($K$). Note that both languages are parameterized by `agents : Type`.

In the theory that follows, it will often be necessary to typecast between the two languages. Indeed, the translation function $t : \mathcal{L}_{K[]} \to \mathcal{L}$ from the language of public announcement logic to the language of epistemic logic is a way of typecasting and is an important part of the completeness proof. However, for typechecking reasons in Lean it is also necessary to define an embedding from the language of epistemic logic to the language of public announcement logic. Thus, in addition to the formal definitions of the languages, we have the following function definition that transforms formulas of type `form` to formulas of type `formPA`.

```
def to_PA : form agents → formPA agents
  | (form.bot)       := formPA.bot
  | (form.var n)     := formPA.var n
  | (form.and φ ψ)   := formPA.and (to_PA φ) (to_PA ψ)
  | (form.impl φ ψ)  := formPA.impl (to_PA φ) (to_PA ψ)
  | (form.box a φ)   := formPA.box a (to_PA φ)
```

Turning to the basic semantics, in epistemic and public announcement logics Kripke frames are again formalized as `structures`. In this case, a `frame` is indexed by the type `agents` and consists of states of some type, a hypothesis that the states are non-empty, and a binary relation for each agent between states.

```
structure frame (agents : Type) :=
  (states : Type)
  (h : nonempty states)
  (rel : agents → states → states → Prop)
```

Next we can define validity. Validity for epistemic logic is specified straightforwardly by cases in Lean as follows.

```
def forces : ∀ f : frame agents,
  (nat → f.states → Prop) → f.states → form agents → Prop
  | f v x form.bot           := false
  | f v x (form.var n)       := v n x
  | f v x (form.and φ ψ)     := (forces f v x φ) ∧ (forces f v x ψ)
  | f v x (form.impl φ ψ)    := (forces f v x φ) → (forces f v x ψ)
  | f v x (form.box a φ)     := ∀ y, f.rel a x y → forces f v y φ
```

Before we can define validity for public announcement logic, we must specify a *frame restriction* in Lean. Recall, the public announcement $[\varphi]$ is a state transformer; it restricts the set of states to only those states where $\varphi$ holds. Thus, a frame restriction is a function that takes a frame `f`, a predicate `P` that the states of the frame must satisfy, a state `s`, and a hypothesis `Ps` that the state `s` indeed satisfies `P`.

```
def frame.restrict (f : frame agents) (P : f.states → Prop)
(s : f.states) (Ps : P s) : frame agents :=
{
  states := { s' : f.states // P s' },
  h := ⟨⟨s, Ps⟩⟩,
  rel := λ a : agents, λ u v, f.rel a u.val v.val
}
```

The resulting structure consists of the subtype of all states `s'` in the original frame that satisfy `P`, a proof that the subtype of states is non-empty, and a binary relation for each agent between states of the subtype.

With frame restrictions now defined, we can use them in our definition of validity for public announcement logic. The definition in Lean is again by cases.

```
def forcesPA : ∀ f : frame agents,
(nat → f.states → Prop) → f.states → formPA agents → Prop
  | f v x bot          := false
  | f v x (var n)      := v n x
  | f v x (and φ ψ)    := (forcesPA f v x φ) ∧ (forcesPA f v x ψ)
  | f v x (impl φ ψ)   := (forcesPA f v x φ) → (forcesPA f v x ψ)
  | f v x (box a φ)    := ∀ y, f.rel a x y → forcesPA f v y φ
  | f v x (update φ ψ) := ∀ h : forcesPA f v x φ,
      forcesPA (f.restrict (λ y, forcesPA f v y φ) x h)
      (λ n u, v n u.val) ⟨x, h⟩ ψ
```

The final clause for update essentially says that whenever $\varphi$ holds at $x$, then $\psi$ holds at $x$ in the restricted frame `f.restrict`, where the restricted frame contains only the set of worlds where $\varphi$ holds.

This concludes the basic syntactic and semantic definitions for public announcement logic that are note-worthy. In the next section, I will discuss the formalization of soundness and completeness results.

### 3.3.2 Soundness and Completeness

The axiom system PA for public announcement logic rests on the axiom system S5 for epistemic logic indexed over agents, which is given by the following definition in Lean.

```
inductive prfS5 : ctx agents → form agents → Prop
  | ax {Γ} {φ} (h : φ ∈ Γ) : prfS5 Γ φ
  | pl1 {Γ} {φ ψ}          : prfS5 Γ φ ⊃ (ψ ⊃ φ)
  | pl2 {Γ} {φ ψ χ}        : prfS5 Γ (φ ⊃ (ψ ⊃ χ)) ⊃ ((φ ⊃ ψ) ⊃ (φ ⊃ χ))
  | pl3 {Γ} {φ ψ}          : prfS5 Γ (¬φ ⊃ ¬ψ) ⊃ (¬φ ⊃ ψ) ⊃ φ
  | pl4 {Γ} {φ ψ}          : prfS5 Γ φ ⊃ (ψ ⊃ (φ & ψ))
  | pl5 {Γ} {φ ψ}          : prfS5 Γ (φ & ψ) ⊃ φ
  | pl6 {Γ} {φ ψ}          : prfS5 Γ (φ & ψ) ⊃ ψ
  | pl7 {Γ} {φ ψ}          : prfS5 Γ (¬φ ⊃ ¬ψ) ⊃ (ψ ⊃ φ)
  | kdist {Γ} {φ ψ} {a}    : prfS5 Γ (K a (φ ⊃ ψ)) ⊃ ((K a φ) ⊃ (K a ψ))
```

```
  | truth {Γ} {φ} {a}      : prfS5 Γ (K a φ) ⊃ φ
  | posintro {Γ} {φ} {a}   : prfS5 Γ (K a φ) ⊃ (K a (K a φ))
  | negintro {Γ} {φ} {a}   : prfS5 Γ (¬(K a φ)) ⊃ (K a ¬(K a φ))
  | mp {Γ} {φ ψ}
    (hpq: prfS5 Γ (φ ⊃ ψ))
    (hp : prfS5 Γ φ)       : prfS5 Γ ψ
  | nec {Γ} {φ} {a}
    (hp: prfS5 Γ φ)        : prfS5 Γ (K a φ)
```

The proof system PA is much the same as the proof system S5, with some additional rules for the update operator that govern how update interacts with other connectives.

```
inductive prfPA : ctxPA agents → formPA agents → Prop
  | ax {Γ} {φ} (h : φ ∈ Γ) : prfPA Γ φ
  | pl1 {Γ} {φ ψ}          : prfPA Γ φ ⊃ (ψ ⊃ φ)
  | pl2 {Γ} {φ ψ χ}        : prfPA Γ (φ ⊃ (ψ ⊃ χ)) ⊃((φ ⊃ ψ) ⊃ (φ ⊃ χ))
  | pl3 {Γ} {φ ψ}          : prfPA Γ (¬φ ⊃ ¬ψ) ⊃ ((¬φ ⊃ ψ) ⊃ φ)
  | pl4 {Γ} {φ ψ}          : prfPA Γ φ ⊃ (ψ ⊃ (φ & ψ))
  | pl5 {Γ} {φ ψ}          : prfPA Γ (φ & ψ) ⊃ φ
  | pl6 {Γ} {φ ψ}          : prfPA Γ (φ & ψ) ⊃ ψ
  | pl7 {Γ} {φ ψ}          : prfPA Γ (¬φ ⊃ ¬ψ) ⊃ (ψ ⊃ φ)
  | kdist {Γ} {φ ψ} {a}    : prfPA Γ (K a (φ ⊃ ψ)) ⊃ ((K a φ) ⊃ (K a ψ))
  | truth {Γ} {φ} {a}      : prfPA Γ (K a φ) ⊃ φ
  | posintro {Γ} {φ} {a}   : prfPA Γ (K a φ) ⊃ (K a (K a φ))
  | negintro {Γ} {φ} {a}   : prfPA Γ (¬(K a φ)) ⊃ (K a ¬(K a φ))
  | atomicbot {Γ} {φ}      : prfPA Γ (U φ ⊥) ↔ (φ ⊃ ⊥)
  | atomicperm {Γ} {φ} {n} : prfPA Γ (U φ (p n)) ↔ (φ ⊃ (p n))
  | announceneg {Γ} {φ ψ}  : prfPA Γ (U φ (¬ψ)) ↔ (φ ⊃ ¬(U φ ψ))
  | announceconj {Γ}
    {φ ψ χ}                : prfPA Γ (U φ (ψ & χ)) ↔ ((U φ ψ) & (U φ χ))
  | announceimp {Γ}
    {φ ψ χ}                : prfPA Γ (U φ (ψ ⊃ χ)) ↔ ((U φ ψ) ⊃ (U φ χ))
  | announceknow {Γ}
    {φ ψ} {a}              : prfPA Γ (U φ (K a ψ)) ↔ (φ ⊃ (K a (U φ ψ)))
  | announcecomp {Γ}
    {φ ψ χ}                : prfPA Γ (U φ (U ψ χ)) ↔ (U (φ & (U φ ψ)) χ)
  | mp {Γ} {φ ψ}
    (hpq: prfPA Γ (φ ⊃ ψ))
    (hp : prfPA Γ φ)       : prfPA Γ ψ
  | nec {Γ} {φ} {a}
    (hp: prfPA Γ φ)        : prfPA Γ (K a φ)
```

With the basic semantics defined and the proof systems S5 and PA defined, we can prove soundness for both proof systems with respect to the class of frames $\mathcal{F}_{eq}$ satisfying an equivalence relation. However, in order to prove soundness for PA, a few lemmas regarding

semantic equivalence are required. These lemmas correspond to those given in proposition 4.22 in chapter 4 of *Dynamic Epistemic Logic* [16].

```
lemma public_announce_var (φ : formPA agents) (f : frame agents)
(v : nat → f.states → Prop) (x : f.states) (n : nat) :
  forcesPA f v x (U φ p n) ↔ forcesPA f v x (φ ⊃ (p n)) := ...

lemma public_announce_conj (φ ψ χ : formPA agents) (f : frame agents)
(v : nat → f.states → Prop) (x : f.states) :
  forcesPA f v x (U φ (ψ & χ)) ↔ forcesPA f v x ((U φ ψ) & (U φ χ)) := ...

lemma public_announce_imp (φ ψ χ : formPA agents) (f : frame agents)
(v : nat → f.states → Prop) (x : f.states) :
  forcesPA f v x (U φ (ψ ⊃ χ)) ↔ forcesPA f v x ((U φ ψ) ⊃ (U φ χ)) := ...

lemma public_announce_neg (φ ψ : formPA agents) (f : frame agents)
(v : nat → f.states → Prop) (x : f.states) :
  forcesPA f v x (U φ (¬ψ)) ↔ forcesPA f v x (φ ⊃ ¬(U φ ψ))  := ...

lemma public_announce_know (φ ψ : formPA agents) (f : frame agents)
(v : nat → f.states → Prop) (x : f.states) (a : agents) :
  forcesPA f v x (U φ (K a ψ)) ↔ forcesPA f v x (φ ⊃ (K a (U φ ψ))) := ...

lemma public_announce_comp (φ ψ χ : formPA agents) (f : frame agents)
  (v : nat → f.states → Prop) (x : f.states) :
  forcesPA f v x (U (φ & U φ ψ) χ) ↔ forcesPA f v x (U φ (U ψ χ)) :=
```

For the proofs of these lemmas, see the file `announcements.lean`. For the proofs of soundness, see theorems `soundnessS5` and `soundnessPA` in appendix C.

---

Turning our attention to completeness, recall from section 2.2 that completeness for system PA with respect to the class of frames $\mathcal{F}_{eq}$ is carried out by translation. That is, we define a translation function $t : \mathcal{L}_{K[]} \to \mathcal{L}_K$ and show that every formula is provably equivalent to its translation. Completeness for PA with respect to $\mathcal{F}_{eq}$ then follows from completeness for S5 with respect to $\mathcal{F}_{eq}$.

To do this in Lean, we must start by defining a complexity measure in order to prove the equivalence of formulas by induction on the complexity of formulas. This complexity measure in Lean is specified by cases below.

```
@[simp] def complexity : formPA agents → nat
  | (⊥)     := 1
  | (p n)   := 1
  | (φ & ψ) := 1 + max (complexity φ) (complexity ψ)
  | (φ ⊃ ψ) := 1 + max (complexity φ) (complexity ψ)
  | (K a φ) := 1 + (complexity φ)
  | (U φ ψ) := (4 + (complexity φ)) * (complexity ψ)
```

With the complexity measure defined, we can give the definition of the translation function $t : \mathcal{L}_{K[]} \to \mathcal{L}_K$. The definition below might be slightly confusing for those unfamiliar with Lean, so a bit of clarification for the reader will be helpful in this case.

This translation function is specified recursively by cases using the equation compiler in Lean. The equation compiler works by attempting to find a well-founded relation on the type being recursed on and automatically attempts to prove the function is well-founded.

However, sometimes one needs to specify a different well-founded relation in order to prove that a recursion is well-founded. To do this, one defines a `has_well_founded` instance. This is a structure with two fields, a relation and a proof that the relation is well-founded.

In this case, we define a `has_well_founded` instance using the complexity measure specified above. The final line in the definition tells the equation compiler to use this relation. For each recursive clause of the definition, we must show that the relation holds by giving Lean a proof that it indeed holds.

```
@[simp] def translate : formPA agents → form agents
  | (formPA.bot)  := form.bot
  | (p n)         := form.var n
  | (φ & ψ)       := have _, from tr1 φ ψ,
     have _, from tr2 φ ψ, form.and (translate φ) (translate ψ)
  | (φ ⊃ ψ)       := have _, from tr1 φ ψ,
     have _, from tr2 φ ψ, form.impl (translate φ) (translate ψ)
  | (K a φ)       := form.box a (translate φ)
  | (U φ ⊥)       := have _, from tr3 φ, translate (φ ⊃ ⊥)
  | (U φ (p n))   := have _, from tr3 φ, translate (φ ⊃ (p n))
  | (U φ (ψ & χ)) := have _, from tr5 φ ψ χ, translate ((U φ ψ) & (U φ χ))
  | (U φ (ψ ⊃ χ)) := have _, from tr5 φ ψ χ, translate ((U φ ψ) ⊃ (U φ χ))
  | (U φ (K a ψ)) := have _, from tr6 φ ψ, translate (φ ⊃ (K a (U φ ψ)))
  | (U φ (U ψ χ)) := have _, from tr7 φ ψ χ, translate (U (φ & (U φ ψ)) χ)
  using_well_founded {rel_tac := λ _ _, `[exact ⟨_, measure_wf complexity⟩]}
```

With the definitions of complexity and the translation function in hand, we can show that every formula is provably equivalent to its translation. For the proof, see theorem `equiv_translation` in appendix C. Completeness of PA with respect to $\mathcal{F}_{eq}$ then follows from completeness of S5 with respect to $\mathcal{F}_{eq}$. For the full proof, again see appendix C.

```
theorem completenessPA {φ : formPA agents} (Γ : ctxPA agents) :
  global_sem_csqPA ∅ equiv_class φ → prfPA ∅ φ := ...
```

This concludes the description of the formalization of modal logic and public announcement logic in Lean. In the next chapter, I will consider related work.

# Chapter 4

# Related Work

The project is not the first formalization of modal logic in Lean, and I would be remiss to not mention how this project benefitted from the prior work of Wu and Goré [20], as well as Bentzen [5]. In particular, Wu and Goré formalized decision procedures for the modal logics K, T and S4 in Lean, and Bentzen formalized the first canonical model construction of completeness for S5 modal logic in Lean, both in 2019.

Although this formalization project also included a canonical model construction of completeness for K, T, S4, and S5 like Bentzen, a great deal of the design decisions differed from that of Bentzen, making both formalizations novel advancements to modal logic in Lean. Additionally, this thesis represent the first steps toward formalizing modal model theoretic arguments and constructions in Lean, and the first formalization of soundness and completeness for dynamic epistemic logic in any proof assistant to date. In the discussion that follows, I will address related formalization work in modal model theory as well as dynamic epistemic logic.

In terms of related formaliztion work in model theory, Fervari et al. [9] proved invariance under bisimulation in the Coq theorem prover in 2019 by formalizing a generalized model update function as well as a notion of bisimulation that is independent with the respect to the model update function. The advantage of this approach is that their notion of bisimulation is applicable to any relation changing logic, such as dynamic logics, hybrid logics, and separation logics.

Additionally, in 2020 Xu and Norrish [21] used Coq to formalize a number of results from the first two chapters of a Blackburn et al. [6]. Their formalization includes the Van Benthem characterization theorem, two versions of Łoś's theorem on the saturation of ultraproduct models, and modal equivalence as bisimilarity between ultrafilter extensions.

In terms of related formalization work in dynamic epistemic logic, most formalizations so far have centered around applications in model-checking and proof theory. First, in 2006 Van Eijck [17] developed a Haskell model-checker called DEMO (Dynamic Epistemic MOdeling) that enables a graphical display of update results and action models. In 2010 Maliković and Čubrilo [12] axiomatized the proof theory of epistemic actions in Coq, specifically in relation to solving knowledge games [12]. Finally, in 2018 Balco, et al. [3] formalized a tool called D.EAK (Display calculus for Epistemic Actions and Knowledge) in Isabelle/HOL, which assists the user in reasoning about propositional sequent calculi.

# Chapter 5

# Conclusions and Future Work

The principal goal of this thesis has been to formalize a variety of results from modal logic within the Lean theorem prover. In particular, I have presented the formalization of both model-theoretic results in definability and undefinability, as well as proof-theoretic results in the soundness and completeness of various systems with respect to their corresponding frame classes, including the basic modal logics K, T, S4, and S5, the epistemic logic S5, and the dynamic epistemic logic of public announcements, PA.

To conclude, I would like to discuss directions for future work. Two areas of interest stand out. The first is the formalization of public announcement logic *with* common knowledge. The second is the formalization of topological semantics with respect to logics of knowledge and belief. I begin with the former.

## 5.1 Future Work

### 5.1.1 PAL with Common Knowledge

The language of public announcement logic with common knowledge is given by the following definition.

**Definition 5.1.1 (The language of PAL with Common Knowledge)** Given a finite set of agents $A$ and a countable set of primitive propositions PROP, the language $\mathcal{L}_{KC[]}$ is defined inductively as follows:

$$\psi := \bot \mid p_n \mid \psi \supset \varphi \mid K_a\varphi \mid C_B\varphi \mid [\varphi]\psi$$

where $a \in A$, $p_n \in$ PROP, and $B \subseteq A$.

Here, common knowledge of $\varphi$ means that everybody in some group $B$ knows that $\varphi$, and everybody knows that everybody knows that $\varphi$, ad infinitum. Semantically, this is represented by the mathematical notion that,

$$C_B\varphi = \bigwedge_{n=0}^{\infty} E_B^n\varphi,$$

where

$$E_B\varphi = \bigwedge_{b\in B} K_b\varphi,$$

and $E_B^0\varphi = \varphi$ and $E_B^{n+1}\varphi = E_B E_B^n\varphi$.[1]

The completeness proof for public announcement logic demonstrated in this thesis utilized a canonical model construction. However, when common knowledge is added to the language, one can show that the logic is non-compact. Thus, a simple canonical model construction cannot be used, and instead a model must be constructed for only a finite fragment of the language. Formalizing the completeness result for public announcement logic with common knowledge would be the next obvious step for further work.

### 5.1.2 Topological Semantics

The next area of interest for future work involves topological semantics. Topology is generally regarded as the study of properties preserved under continuous deformations, such as stretching, bending, or shrinking, but not tearing or shredding.

Why would one want to study modal logic through the lens of topology? Topological spaces are interesting with respect to modal logic because they provide a framework for representing evidence and its relationship to knowledge and belief. Much of this thesis centered around the epistemic interpretation of modalities, so generalizing results from relational semantics to topological semantics would enable the formalization of many of the new and exciting advancements in formal epistemology.

What exactly is topological semantics, though? The definition of topological models – and a corresponding notion of topological validity – are defined as follows.

**Definition 5.1.2 (Topological Model)** A topological model is a topological space $(X, \mathcal{T})$ together with a valuation $v : \text{PROP} \to 2^X$.

**Definition 5.1.3 (Topological Validity)** Truth in a model $M = (X, \mathcal{T}, v)$ is defined as

$$\begin{aligned}
[\![p]\!]_M &= v(p) \\
[\![\neg\varphi]\!]_M &= X \setminus [\![\varphi]\!]_M \\
[\![\varphi \wedge \psi]\!]_M &= [\![\varphi]\!]_M \cap [\![\psi]\!]_M \\
[\![\Box\varphi]\!]_M &= int([\![\varphi]\!]_M)
\end{aligned}$$

where $int([\![\varphi]\!]_M)$ denotes the topological interior of the set $[\![\varphi]\!]$.

A primary area of interest for me is to formalize is the completeness of S4 modal logic with respect to the class of all topological spaces.

**Theorem 5.1.4 (Completeness)** The system S4 is sound and complete with respect to the class of all topological spaces.

The connection between topological spaces and S4 modal logic dates back to Alfred Tarski in the 1940s [14], but it still has not been formalized in any theorem prover. Lean would be naturally suited to this formalization project due to mathlib's extensive topology library.

[1] For instance, $E_B^2\varphi$ means that everybody knows that everybody knows that $\varphi$.

# Chapter 6

# Acknowledgements

I cannot begin to express my thanks to Jeremy Avigad, who has supervised this thesis in its entirety. First and foremost, I am thankful for his unparalleled expertise and instruction not only in Lean and interactive theorem proving, but in all things logic-related. Without his continual guidance, this thesis would not have been possible. I would also like to thank him for his enduring support as an advisor. There have been many times (especially during the COVID-19 pandemic, but also during my initial adjustment to academic life at Carnegie Mellon) that Jeremy has helped me stay motivated and keep going. Thank you, Jeremy, for your patient help and your constant support.

I would also like to extend my sincere gratitude to Adam Bjorndahl and Wilfried Sieg. I am most thankful for Adam's mastery in the subjects of modal logic and formal epistemology, without which I would not have gained the knowledge required to complete this thesis. Likewise, I am thankful for Wilfried's expertise in mathematical logic and the philosophy of mathematics, which has provided me with a depth of understanding toward both the history and practices of logic that has enabled me to understand the field more broadly. Finally, I would like to thank them for agreeing to sit on my thesis committee, and for their support as mentors at Carnegie Mellon.

Many thanks to friends, TAs, colleagues, and alumni in the Carnegie Mellon community who discussed various aspects of interactive theorem proving, modal logic, and formal epistemology with me; Dave Vitale, Fernando Larrain, Jacob Neumann, Mario Carniero, and Seul Baek come to mind. I am also grateful to Conny Kneiling, Matt Thorpe, and Will Gunther for their moral support and encouragement.

Finally, my accomplishments would not have been possible without the love and support of my parents. Special thanks to Tom and Sue.

# Chapter 7

# Bibliography


[1] Hajnal Andréka, Johan Van Benthem, and Istvan Németi. Back and forth between modal logic and classical logic. 1995.

[2] Jeremy Avigad, Leonardo de Moura, and Soonho Kong. *Theorem Proving in Lean*, volume 3.4.0.

[3] Samuel Balco, Sabine Frittella, Giuseppe Greco, Alexander Kurz, and Alessandra Palmigiano. Software tool support for modular reasoning in modal logics of actions. In *International Conference on Interactive Theorem Proving*, pages 48–67. Springer, 2018.

[4] Alexandru Baltag and Bryan Renne. Dynamic Epistemic Logic. In Edward N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, winter 2016 edition, 2016.

[5] Bruno Bentzen. A Henkin-style completeness proof for the modal logic S5. *arXiv preprint arXiv:1910.01697*, 2019.

[6] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*, volume 53. Cambridge University Press, 2002.

[7] Leah Crane. Google has created a maths AI that has already proved 1200 theorems. *New Scientist*, April 2019.

[8] Ernest Davis and Leora Morgenstern. Epistemic logic and its applications: Tutorial notes. In *International Joint Conferences on Artificial Intelligence*, volume 93. Citeseer, 1983.

[9] Raul Fervari, Francisco Trucco, and Beta Ziliani. Mechanizing bisimulation theorems for relation-changing logics in coq. In *International Workshop on Dynamic Logic*, pages 3–18. Springer, 2019.

[10] Kevin Hartnett. Building the mathematical library of the future. *Quanta Magazine*, Oct 2020.

[11] Clarence Irving Lewis. *The place of intuition in knowledge*. PhD thesis, Harvard University, 1910.

[12] Marko Maliković and Mirko Čubrilo. Reasoning about epistemic actions and knowledge in multi-agent systems using coq. *Comput. Technol. Appl.*, 2(8):616–627, 2011.

[13] The mathlib Community. The lean mathematical library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, page 367–381, New York, NY, USA, 2020. Association for Computing Machinery.

[14] John Charles Chenoweth McKinsey and Alfred Tarski. The algebra of topology. *Annals of mathematics*, pages 141–191, 1944.

[15] Robert Stalnaker. On logics of knowledge and belief. *Philosophical studies*, 128(1):169–199, 2006.

[16] Hans Van Ditmarsch, Wiebe van Der Hoek, and Barteld Kooi. *Dynamic epistemic logic*, volume 337. Springer Science & Business Media, 2007.

[17] Jan van Eijck. a demo of epistemic modelling. In *Interactive Logic. Selected Papers from the 7th Augustus de Morgan Workshop, London*, volume 1, pages 303–362, 2007.

[18] Moshe Y Vardi. Why is modal logic so robustly decidable? Technical report, 1997.

[19] Mathias Winckel and Ralph Matthes. Formalization of a dynamic logic for graph transformation in the Coq proof assistant.

[20] Minchao Wu and Rajeev Goré. Verified decision procedures for modal logics. In *10th International Conference on Interactive Theorem Proving (ITP 2019)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2019.

[21] Yiming Xu and Michael Norrish. Mechanised modal model theory. In *International Joint Conference on Automated Reasoning*, pages 518–533. Springer, 2020.

# Appendices

# Appendix A

# Invariance Proofs

### A.0.1 Invariance Under Disjoint Union

```
variables {α β : Type}

def frame.rel.dunion (r1 : α → α → Prop) (r2 : β → β → Prop) :
(sum α β) → (sum α β) → Prop
  | (sum.inl a1) (sum.inl a2) := r1 a1 a2
  | (sum.inr b1) (sum.inr b2) := r2 b1 b2
  | _ _                       := false


def val_dunion {f1 f2 : frame} (v1 : nat → f1.states → Prop)
(v2 : nat → f2.states → Prop) : nat → (sum f1.states f2.states) → Prop
  | n (sum.inl x1) := v1 n x1
  | n (sum.inr x2) := v2 n x2


def frame.dunion (f1 f2 : frame) : frame :=
{
  states := sum f1.states f2.states,
  h      := ⟨sum.inl f1.h.default⟩,
  rel    := frame.rel.dunion (f1.rel) (f2.rel)
}
```

```
theorem invariance_dunion (f1 f2 : frame) (v1 : nat → f1.states → Prop)
(v2 : nat → f2.states → Prop) (x1 : f1.states) :
  theory_at f1 v1 x1 = theory_at (f1.dunion f2) (val_dunion v1 v2) (sum.inl x1) :=
begin
ext φ, revert x1, induction φ,
repeat {intro x1},
repeat {split},
repeat {intro h, exact h},
{intro h, split, exact (ih_φ x1).mp h.left,
exact (ih_ψ x1).mp h.right},
{intro h, split, exact (ih_φ x1).mpr h.left,
exact (ih_ψ x1).mpr h.right},
{intros h h1, exact (ih_ψ x1).mp (h ((ih_φ x1).mpr h1))},
{intros h h1, exact (ih_ψ x1).mpr (h ((ih_φ x1).mp h1))},
{intros h s h1, cases s, exact (ih_φ s).mp ((h s) h1),
exact false.elim h1},
{intros h s h1, exact (ih_φ s).mpr ((h (sum.inl s)) h1)}
end
```

### A.0.2 Invariance Under Generated Submodels

The reader is encouraged to refer to the file `paths.lean` for definitions and proofs related to reachability.

```
def frame.gen_subframe (f : frame) (x : f.states) : frame :=
{
  states := { y // reachable (f.rel) x y},
  h      := ⟨x, ref_close x (f.rel)⟩,
  rel    := λ x1 x2, (f.rel) x1.val x2.val
}


def val_gen_subframe (f : frame) (x : f.states)
(v : nat → f.states → Prop) : nat → (frame.gen_subframe f x).states → Prop :=
  λ n, λ y, v n y.val


theorem invariance_gen_submodel (f : frame) (v : nat → f.states → Prop)
(x : f.states) (y : (frame.gen_subframe f x).states) :
  theory_at f v y.val = theory_at (f.gen_subframe x) (val_gen_subframe f x v) y :=
begin
ext φ, revert y, induction φ,
repeat {intro y},
repeat {split},
repeat {intro h1, exact h1},
{intro h1, split, exact (ih_φ y).mp h1.left,
exact (ih_ψ y).mp h1.right},
{intro h1, split, exact (ih_φ y).mpr h1.left,
exact (ih_ψ y).mpr h1.right},
{intros h1 h2, exact (ih_ψ y).mp (h1 ((ih_φ y).mpr h2))},
{intros h1 h2, exact (ih_ψ y).mpr (h1 ((ih_φ y).mp h2))},
{intros h1 z h2, exact (ih_φ z).mp ((h1 z.val) h2)},
{intros h1 z h2,
have h3 := reach_right x y.1 z f.rel (and.intro y.2 h2),
exact (ih_φ ⟨z, h3⟩).mpr ((h1 ⟨z, h3⟩) h2)}
end
```

### A.0.3 Invariance Under Surjective Bounded Morphisms

```
open function

def is_bddmorphism {f1 f2 : frame} (g : f1.states → f2.states) :=
  ∀ x1 : f1.states, forth (λ x1 x2, (g x1) = x2) x1 (g x1)
  ∧ back (λ x1 x2, (g x1) = x2) x1 (g x1)


def is_surjbddmorphism {f1 f2 : frame} (g : f1.states → f2.states) :=
  surjective g ∧ is_bddmorphism g


theorem pull_back {f1 f2 : frame} (g : f1.states → f2.states)
(h : is_surjbddmorphism g) : ∀φ, ¬ f_valid φ f2 → ¬ f_valid φ f1 :=
begin
intros φ h1, rw f_valid at *, push_neg at h1, push_neg,
cases h1 with v2 h1, cases h1 with y h1,
let v1 := (λ n x, v2 n (g x)), use v1,
cases h with hl hr, cases hl y with x hl,
existsi (x : f1.states),
have h3 := invariance_bisim v1 v2 (λ x y, g x = y),
have h4 : is_bisimulation v1 v2 (λ (x : f1.states) (y : f2.states), g x = y),
{intros x1 x2 h2, split,
{intro n, split, intro h, subst h2, apply h, intro h, subst h2, apply h},
split,
have h5 : forth (λ (x1 : f1.states) (x2 : f2.states), g x1 = x2) x1 x2,
from eq.subst h2 (hr x1).left, exact h5,
have h5 : back (λ (x1 : f1.states) (x2 : f2.states), g x1 = x2) x1 x2,
from eq.subst h2 (hr x1).right, exact h5},
specialize h3 h4 x y hl, intro h2, apply h1,
rw set.subset.antisymm_iff at h3, cases h3,
rw set.subset_def at h3_left, exact h3_left φ h2
end


theorem invariance_pull_back (F : set (frame)) {f1 f2 : frame}
(h1 : f1 ∈ F) (h2 : f2 ∉ F) :
  (∃ g : f1.states → f2.states, is_surjbddmorphism g) → undefinable F :=
begin
intro h, cases h with g h,
intro φ, by_contradiction h3,
have h4 := h3 f1, specialize h3 f2,
rw ←not_iff_not at h3,
exact (pull_back g h φ (h3.mp h2)) (h4.mp h1)
end
```

### A.0.4 Invariance Under Bisimulation

```
def base {f1 f2 : frame} (v1 : nat → f1.states → Prop)
(v2 : nat → f2.states → Prop) (x1 : f1.states) (x2 : f2.states) :=
  ∀ n, v1 n x1 ↔ v2 n x2

def forth {f1 f2 : frame} (bisim : f1.states → f2.states → Prop)
(x1 : f1.states) (x2 : f2.states) :=
  ∀ y1, f1.rel x1 y1 → (∃ y2 : f2.states, f2.rel x2 y2 ∧ bisim y1 y2)

def back {f1 f2 : frame} (bisim : f1.states → f2.states → Prop)
(x1 : f1.states) (x2 : f2.states) :=
  ∀ y2, f2.rel x2 y2 → (∃ y1 : f1.states, f1.rel x1 y1 ∧ bisim y1 y2)

def is_bisimulation {f1 f2 : frame} (v1 : nat → f1.states → Prop)
(v2 : nat → f2.states → Prop)
(bisim : f1.states → f2.states → Prop) :=
  ∀ x1 x2, bisim x1 x2 → (base v1 v2 x1 x2 ∧ forth bisim x1 x2
  ∧ back bisim x1 x2)

theorem invariance_bisim {f1 f2 : frame} (v1 : nat → f1.states → Prop)
(v2 : nat → f2.states → Prop) (bisim : f1.states → f2.states → Prop)
(h : is_bisimulation v1 v2 bisim) (x1 : f1.states) (x2 : f2.states) :
  bisim x1 x2 → theory_at f1 v1 x1 = theory_at f2 v2 x2 :=
begin
intro h1, ext φ, revert x1 x2, induction φ,
repeat {intros x1 x2 h1},
{split, repeat {intro h2, exact h2}},
{specialize h x1 x2 h1, split,
intro h2, exact (h.left φ).mp h2,
intro h2, exact (h.left φ).mpr h2},
{specialize ih_φ x1 x2 h1, specialize ih_ψ x1 x2 h1, split,
intro h2, split, exact ih_φ.mp h2.left, exact ih_ψ.mp h2.right,
intro h2, split, exact ih_φ.mpr h2.left, exact ih_ψ.mpr h2.right},
{specialize ih_φ x1 x2 h1, specialize ih_ψ x1 x2 h1, split,
intros h2 h3, exact ih_ψ.mp (h2 (ih_φ.mpr h3)),
intros h2 h3, exact ih_ψ.mpr (h2 (ih_φ.mp h3))},
{specialize h x1 x2 h1, cases h with h4 h5, cases h5 with h5 h6, split,
{intros h2 y2 h3, specialize h6 y2 h3, cases h6 with y1 h6,
cases h6 with h6 h7, exact (ih_φ y1 y2 h7).mp (h2 y1 h6)},
{intros h2 y1 h3, specialize h5 y1 h3, cases h5 with y2 h5,
cases h5 with h5 h7, exact (ih_φ y1 y2 h7).mpr (h2 y2 h5)}}
end
```

# Appendix B

# Modal Logic Proofs

### B.0.1 Soundness for System K

```
theorem soundness (AX : ctx) (F : set (frame)) (φ : form) :
  prfK AX φ → global_sem_csq AX F φ :=
begin
intro h,
induction h,
{intros f h v h1 x,
exact (h1 h_φ x) h_h},
{intros f h2 v h3 x h4 h5, exact h4},
{intros f h2 v h3 x h4 h5 h6, apply h4,
exact h6, apply h5, exact h6},
{intros f h2 v h3 x h4 h5,
by_contradiction h6,
exact (h4 h6) (h5 h6)},
{intros f h2 v h3 x h4 h5,
exact and.intro h4 h5},
{intros f h2 v h3 x h4, exact h4.left},
{intros f h2 v h3 x h4, exact h4.right},
{intros f h2 v h3 x h4 h5,
repeat {rw forces at h4},
repeat {rw imp_false at h4},
rw not_imp_not at h4,
exact h4 h5},
{intros f h2 v h3 x h4 h5 x' h6,
exact h4 x' h6 (h5 x' h6)},
{intros f h2 v h3 x,
exact (h_ih_hpq f h2 v h3 x) (h_ih_hp f h2 v h3 x)},
{intros f h2 v h3 x y h4,
exact h_ih f h2 v h3 y},
end
```

### B.0.2 Soundness for System **T**

```
lemma soundnesshelper {Γ : ctx} {φ : form} {C : set (frame)} :
  prfK Γ φ → (∀ ψ ∈ Γ, F_valid ψ C) → global_sem_csq Γ C φ :=
begin
intros h1 h2 f h3 v h4, induction h1,
{intro x, exact h2 h1_φ h1_h f h3 v x},
{intros x h4 h5, exact h4},
{intros x h4 h5 h6, exact (h4 h6) (h5 h6)},
{intros x h3 h4, by_contradiction h5, specialize h3 h5,
rw ←not_forces_imp at h3, exact h3 (h4 h5)},
{intros x h4 h5, exact and.intro h4 h5},
{intros x h4, exact h4.left},
{intros x h4, exact h4.right},
{intros x h4 h5, repeat {rw forces at h4},
repeat {rw imp_false at h4},
rw not_imp_not at h4, exact h4 h5},
{intros x h3 h4, intros x' h5,
exact (h3 x' h5) (h4 x' h5)},
{intro x, exact h1_ih_hpq h2 h4 x (h1_ih_hp h2 h4 x)},
{intros x y h3, apply h1_ih, exact h2, exact h4}
end


lemma inclusion_valid {C C' : set (frame)} :
  ∀ ψ, C ⊆ C' → F_valid ψ C' → F_valid ψ C :=
begin
intros φ h1 h2 f h3 v x,
exact h2 f (set.mem_of_subset_of_mem h1 h3) v x
end


lemma T_helper : ∀ φ ∈ T_axioms, F_valid φ ref_class :=
begin
intros φ h1 f h2 v x,
cases h1 with ψ h1, subst h1,
apply ref_helper, exact h2
end


theorem T_soundness (φ : form) :
  prfK T_axioms φ → global_sem_csq T_axioms ref_class φ :=
begin
intro h, apply soundnesshelper, apply h, apply T_helper
end
```

### B.0.3 Soundness for System **S4**

```
lemma S4_helper : ∀ φ ∈ S4_axioms, F_valid φ ref_trans_class :=
begin
intros φ h1 f h2 v x,
cases h2 with h2l h2r,
cases h1 with h1 h3,
{apply T_helper, exact h1, exact h2l},
{cases h3 with ψ h3, subst h3,
apply trans_helper, exact h2r}
end


theorem S4_soundness (φ : form) :
  prfK S4_axioms φ → global_sem_csq S4_axioms ref_trans_class φ :=
begin
intro h, apply soundnesshelper, apply h, apply S4_helper
end
```

### B.0.4 Soundness for System **S5**

```
lemma S5_helper : ∀ φ ∈ S5_axioms, F_valid φ equiv_class :=
begin
intros φ h1 f h2 v x,
cases h2 with h2l h2r,
cases h2r with h2r h2rr,
cases h1 with h1 h3,
{apply T_helper, exact h1,
exact h2l},
{cases h3 with ψ h3, dsimp at h3,
subst h3, apply euclid_helper,
intros x y z h1 h2,
exact h2rr (h2r h1) h2}
end


theorem S5_soundness (φ : form) :
  prfK S5_axioms φ → global_sem_csq S5_axioms equiv_class φ :=
begin
intro h, apply soundnesshelper, apply h, apply S5_helper
end
```

### B.0.5 Completeness for System **K**

```
open zorn

lemma lindenbaum (AX Γ : ctx) (hax : ax_consist AX Γ) :
  ∃ Γ', max_ax_consist AX Γ' ∧ Γ ⊆ Γ' :=
begin
let P := { Γ'' | Γ'' ⊇ Γ ∧ ax_consist AX Γ''},
have h : ∀ c ⊆ P, chain (⊆) c → c.nonempty → ∃ ub ∈ P,
  ∀ s ∈ c, s ⊆ ub,
{intros c h2 h3 h4, use ⋃₀(c),
have h5 := lindenhelper c h4 h3,
repeat {split},
cases h4 with Γ'' h4,
have h6 := set.mem_of_mem_of_subset h4 h2,
cases h6 with h6 h7,
apply set.subset_sUnion_of_subset c Γ'' h6 h4,
intros L h6,
cases h5 L h6 with m h5,
cases h5 with h7 h5,
cases (set.mem_of_mem_of_subset h7 h2) with h8 h9,
apply h9, exact h5,
intros s h7, exact set.subset_sUnion_of_mem h7},
have h1 : Γ ∈ P,
split, exact set.subset.rfl, exact hax,
cases zorn_subset₀ P h Γ h1 with Γ' h2,
cases h2 with h2 h3, cases h3 with h3 h4,
use Γ', split, rw max_equiv, split, apply h2.2,
intros m h5 h6, symmetry, apply h4 m, split,
apply set.subset.trans h2.1 h6,
exact h5, exact h6, apply h2.1
end

lemma max_ax_exists (AX : ctx) (hax : sem_cons AX) :
  ∃ Γ : ctx, max_ax_consist AX Γ :=
begin
have h1 : ax_consist AX ∅,
{intro L, intro h2, rw fin_ax_consist,
have h3 := listempty h2, have this : ∅ = ∅, refl,
specialize h3 this, subst h3, by_contradiction h4,
apply nprfalse AX hax, exact mp dne h4},
have h2 := lindenbaum AX ∅ h1,
cases h2 with Γ h2, cases h2 with h2 h3, existsi (Γ : ctx), apply h2
end
```

```
lemma existence (AX : ctx) (hax : sem_cons AX)
(xΓ : (canonical AX).states) :
  ∀ φ, ◇φ ∈ xΓ.val ↔ ∃ yΔ : (canonical AX).states, φ ∈ yΔ.val
  ∧ (canonical AX).rel xΓ yΔ :=
begin
intro φ, split, intro h1,
let Γbox : ctx := {ψ : form | □ψ ∈ xΓ.val},
have h1 : ax_consist AX (Γbox ∪ {φ}),
{by_contradiction h2, simp at h2,
have h3 := five AX Γbox φ h2,
cases h3 with L h3, cases h3 with h3 h4,
have h5 := cut fin_conj_boxn (mp kdist (nec h4)),
have h6 := exercise1,
have h7 : ∀ ψ ∈ (list.map □L), ψ ∈ xΓ.1,
intros ψ h8, simp at *, cases h8 with a h8, cases h8 with h8l h8r,
subst h8r, exact h3 a h8l, specialize h6 xΓ.2 h7 h5,
have h8 := (six AX xΓ.1 (max_imp_ax xΓ.2)).mp xΓ.2 (¬φ).box,
cases h8 with h8l h8r, simp at *,
exact absurd h1 (h8r h6)},
have h2 := lindenbaum AX (Γbox ∪ {φ}) h1,
cases h2 with Δ h2, cases h2 with h2 h3,
let xΔ : (canonical AX).states := ⟨Δ, h2⟩,
existsi (xΔ : (canonical AX).states),
have h5 := set.union_subset_iff.mp h3,
cases h5, split, simp at h5_right, exact h5_right,
have h3 : ∀ φ : form, □φ ∈ xΓ.val → φ ∈ xΔ.val,
intros ψ h4, apply h5_left, exact h4, exact h3,
simp at *, intros yΔ h1 h2, by_contradiction h3,
have h4 := (max_notiff AX xΓ.1 xΓ.2 (◇φ)).mp h3,
have h5 := (max_dn AX xΓ.1 xΓ.2 (□¬φ)).mpr h4,
have h6 := (max_notiff AX yΔ.1 yΔ.2 φ).mpr (h2 (¬φ) h5),
exact absurd h1 h6
end

lemma truth (AX : ctx) (hax : sem_cons AX) (xΓ : (canonical AX).states) :
  ∀ φ, forces (canonical AX) (val_canonical AX) xΓ φ ↔ (φ ∈ xΓ.val) :=
begin
intro φ, induction φ generalizing xΓ,
split, intro h1, exact false.elim h1,
intro h1,
have h2 := xΓ.2,
cases h2,
specialize h2_left [⊥],
simp at *,
exact absurd not_contra (h2_left h1),
```

```
repeat {rw forces, rw val_canonical},
split, intro h1, cases h1 with h1 h2,
exact max_conj_1 xΓ.2 (and.intro ((ih_φ xΓ).mp h1) ((ih_ψ xΓ).mp h2)),
intro h1, split,
apply (ih_φ xΓ).mpr, exact max_conj_2 xΓ.2 h1,
apply (ih_ψ xΓ).mpr, exact max_conj_3 xΓ.2 h1,
split,
intro h1,
apply max_imp_1 xΓ.2,
intro h2,
exact (ih_ψ xΓ).mp (h1 ((ih_φ xΓ).mpr h2)),
intros h1 h2,
apply (ih_ψ xΓ).mpr,
exact max_imp_2 xΓ.2 h1 ((ih_φ xΓ).mp h2),
split, intros h1,
by_contradiction h2,
have h4 := (existence AX hax xΓ (¬φ)).mp,
have h5 := max_boxdn AX xΓ.1 xΓ.2 φ
 ((max_notiff AX xΓ.1 xΓ.2 φ.box).mp h2),
cases h4 h5 with xΔ h4, cases h4 with h4 h6,
have h7 := max_notiff AX xΔ.1 xΔ.2 φ,
cases h7 with h7l h7r,
exact absurd ((ih_φ xΔ).mp (h1 xΔ h6)) (h7r h4),
intros h1 xΔ h2,
apply (ih_φ xΔ).mpr, exact h2 φ h1,
end

theorem completeness (AX : ctx) (hax : sem_cons AX) (φ : form) :
  global_sem_csq AX all_class φ → prfK AX φ :=
begin
rw ← not_imp_not, intro h1,
have h2 := comphelper AX φ hax h1,
have h3 := lindenbaum AX {¬φ} h2,
simp at *,
cases h3 with Γ' h3, cases h3 with h3 h4,
rw global_sem_csq,
push_neg,
let f := canonical, use f AX,
split,
trivial,
let v := val_canonical, use v AX,
let xΓ' : (f AX).states := ⟨Γ', h3⟩,
split,
exact forcesAX AX hax,
use xΓ',
```

```
have h5 := truth AX hax xΓ' ¬φ,
cases h5 with h5 h6,
have h7 := not_forces_imp (f AX) (v AX) xΓ' φ,
cases h7 with h7 h8, apply h8, apply h6, exact h4
end
```

### B.0.6 Completeness for System **T**

```
lemma T_reflexive : T_canonical ∈ ref_class :=
begin
intros x φ h1,
have h2 : (∀ ψ ∈ [□φ], ψ ∈ x.1) →
  prfK T_axioms (fin_conj [□φ] ⊃ φ) → φ ∈ x.1,
  from exercise1 x.2, simp at *,
have h3 : prfK T_axioms (fin_conj [□φ] ⊃ φ),
{repeat {rw fin_conj},
have h4 : prfK T_axioms (□φ ⊃ φ),
{refine ax _, rw T_axioms, simp},
exact cut (mp pl5 phi_and_true) h4},
exact h2 h1 h3
end

theorem T_completeness (φ : form) :
  global_sem_csq T_axioms ref_class φ → prfK T_axioms φ :=
begin
rw ←not_imp_not,
intro h1,
rw global_sem_csq,
push_neg,
let f := T_canonical, use f,
split,
exact T_reflexive,
let v := val_canonical, use (@v T_axioms sem_consT),
split,
exact forcesAX T_axioms sem_consT,
have h4 := lindenbaum T_axioms {¬φ} (comphelper T_axioms φ sem_consT h1),
simp at *,
cases h4 with Γ' h4, cases h4 with h4 h5,
let xΓ : f.states := ⟨Γ', h4⟩,
use xΓ,
have h6 := truth T_axioms sem_consT xΓ ¬φ,
cases h6 with h6 h7,
have h8 := not_forces_imp f (@v T_axioms sem_consT) xΓ φ,
cases h8 with h8 h9, apply h9, apply h7, exact h5
end
```

### B.0.7 Completeness for System **S4**

```
lemma S4_reftrans : S4_canonical ∈ ref_trans_class :=
begin
split,
intros x φ h1,
have h2 : (∀ ψ ∈ [□φ], ψ ∈ x.1) →
  prfK TS4_axioms (fin_conj [□φ] ⊃ φ) → φ ∈ x.1,
  from exercise1 x.2, simp at *,
have h3 : prfK S4_axioms (fin_conj [□φ] ⊃ φ),
{repeat {rw fin_conj},
have h4 : prfK S4_axioms (□φ ⊃ φ),
{refine ax _, rw S4_axioms, simp, rw T_axioms, simp},
exact cut (mp pl5 phi_and_true) h4},
exact h2 h1 h3,
intros x y z h1 h2 φ h3, apply h2 φ,
apply h1 (□φ),
have h4 : prfK S4_axioms (fin_conj [□φ] ⊃ □□φ),
{repeat {rw fin_conj},
have h5 : prfK S4_axioms (□φ ⊃ □□φ),
{refine ax _, rw S4_axioms, simp},
exact cut (mp pl5 phi_and_true) h5},
have h6 : (∀ ψ ∈ [□φ], ψ ∈ x.1) →
  prfK S4_axioms (fin_conj [□φ] ⊃ □□φ) → (□□φ) ∈ x.1,
  from exercise1 x.2, simp at *,
exact h6 h3 h4
end

theorem S4_completeness (φ : form) :
  global_sem_csq S4_axioms ref_trans_class φ → prfK S4_axioms φ :=
begin
rw ←not_imp_not, intro h1, rw global_sem_csq, push_neg,
let f := S4_canonical, use f,
split, exact S4_reftrans,
let v := val_canonical, use (@v S4_axioms sem_consS4),
split, exact forcesAX S4_axioms sem_consS4,
have h4 := lindenbaum S4_axioms {¬φ} (comphelper S4_axioms φ sem_consS4 h1),
simp at *, cases h4 with Γ' h4, cases h4 with h4 h5,
let xΓ : f.states := ⟨Γ', h4⟩, use xΓ,
have h6 := truth S4_axioms sem_consS4 xΓ ¬φ,
cases h6 with h6 h7,
have h8 := not_forces_imp f (@v S4_axioms sem_consS4) xΓ φ,
cases h8 with h8 h9, apply h9, apply h7, exact h5
end
```

### B.0.8 Completeness for System **S5**

```
lemma S5_equiv : S5_canonical ∈ equiv_class :=
begin
rw equiv_ref_euclid, split, intros x φ h1,
have h2 : (∀ ψ ∈ [□φ], ψ ∈ x.1) →
  prfK TS4_axioms (fin_conj [□φ] ⊃ φ) → φ ∈ x.1,
  from exercise1 x.2, simp at *,
have h3 : prfK S5_axioms (fin_conj [□φ] ⊃ φ),
{repeat {rw fin_conj},
have h4 : prfK S5_axioms (□φ ⊃ φ),
{refine ax _, rw S5_axioms, simp, rw T_axioms, simp},
exact cut (mp pl5 phi_and_true) h4},
exact h2 h1 h3, intros x y z h1 h2 φ h3, apply h2 φ, clear h2,
have h2 : prfK S5_axioms (◊(¬φ) ⊃ □(◊¬φ)),
{refine ax _, rw S5_axioms, simp},
have h4 : prfK S5_axioms (◊(□φ) ⊃ □φ),
  from mp euclid_dual h2,
have h5 : (∀ ψ ∈ [◊(□φ)], ψ ∈ x.1) →
  prfK S5_axioms (fin_conj [◊(□φ)] ⊃ □φ) → □φ ∈ x.1,
  from exercise1 x.2, simp at *,
apply h5, by_contradiction h6,
have h7 := (max_notiff S5_axioms x.1 x.2 (¬(¬φ.box).box)).mp h6,
have h8 := (max_dn S5_axioms x.1 x.2 ((¬φ.box).box)).mpr h7,
have h9 := (max_notiff S5_axioms y.1 y.2 (φ.box)).mpr (h1 (¬φ.box) h8),
exact absurd h3 h9,
exact (cut (mp pl5 phi_and_true) h4)
end

theorem S5_completeness (φ : form) :
  global_sem_csq S5_axioms equiv_class φ → prfK S5_axioms φ :=
begin
rw ←not_imp_not, intro h1, rw global_sem_csq, push_neg,
let f := S5_canonical, use f, split, exact S5_equiv,
let v := val_canonical, use (@v S5_axioms sem_consS5),
split, exact forcesAX S5_axioms sem_consS5,
have h4 := lindenbaum S5_axioms {¬φ}
  (comphelper S5_axioms φ sem_consS5 h1),
simp at *, cases h4 with Γ' h4, cases h4 with h4 h5,
let xΓ : f.states := ⟨Γ', h4⟩, use xΓ,
have h6 := truth S5_axioms sem_consS5 xΓ ¬φ,
cases h6 with h6 h7,
have h8 := not_forces_imp f (@v S5_axioms sem_consS5) xΓ φ,
cases h8 with h8 h9, apply h9, apply h7, exact h5
end
```

# Appendix C

# Dynamic Epistemic Logic Proofs

### C.0.1 Soundness for System **S5** (Epistemic Logic)

```
theorem soundnessS5 {Γ : ctx agents} {φ : form agents} :
  prfS5 Γ φ → global_sem_csq Γ equiv_class φ :=
begin
intros h1 f h2 v h3 x,
induction h1 generalizing x,
{exact h3 φ x h1},
{intros h4 h5, exact h4},
{intros h4 h5 h6, exact (h4 h6) (h5 h6)},
{intros h1 h4, by_contradiction h5, exact (h1 h5) (h4 h5)},
{intros h4 h5, exact and.intro h4 h5},
{intros h4, exact h4.left},
{intros h4, exact h4.right},
{intros h1 h4, repeat {rw forces at h1}, repeat {rw imp_false at h1},
rw not_imp_not at h1, exact h1 h4},
{intros h3 h4, simp [forces] at *, intros x' h5, exact (h3 x' h5) (h4 x' h5)},
{intros h1, apply h1, exact (h2 a).left x},
{intros h1 y h4 z h5, apply h1 z,
cases h2 a with h2l h2r, cases h2r with h2r h2rr, exact h2rr h4 h5},
{intros h1 y h5 h4, apply h1, intros z h6, apply h4 z,
cases h2 a with h2l h2r, cases h2r with h2r h2rr, exact h2rr (h2r h5) h6},
{exact ih_hpq h3 x (ih_hp h3 x)},
{intros y h1, exact ih h3 y}
end
```

### C.0.2 Soundness for System **PA** (Public Announcement Logic)

```
theorem soundnessPA {Γ : ctxPA agents} {φ : formPA agents} :
  prfPA Γ φ → global_sem_csqPA Γ equiv_class φ :=
begin
intros h1 f h2 v h3 x,
induction h1 generalizing x,
{exact h3 φ x h1},
{intros h4 h5, exact h4},
{intros h4 h5 h6, exact (h4 h6) (h5 h6)},
{intros h1 h4, by_contradiction h5, exact (h1 h5) (h4 h5)},
{intros h4 h5, exact and.intro h4 h5},
{intros h4, exact h4.left},
{intros h4, exact h4.right},
{intros h1 h4, repeat {rw forcesPA at h1}, repeat {rw imp_false at h1},
rw not_imp_not at h1, exact h1 h4},
{intros h3 h4, simp [forcesPA] at *, intros x' h5, exact (h3 x' h5) (h4 x' h5)},
{intros h1, apply h1, exact (h2 a).left x},
{intros h1 y h4 z h5, apply h1 z,
cases h2 a with h2l h2r, cases h2r with h2r h2rr, exact h2rr h4 h5},
{intros h1 y h5 h4, apply h1, intros z h6, apply h4 z,
cases h2 a with h2l h2r, cases h2r with h2r h2rr, exact h2rr (h2r h5) h6},
{exact ih_hpq h3 x (ih_hp h3 x)},
{intros y h1, exact ih h3 y},
{split, repeat {intros h1 h4, apply h1 h4}},
{split, rw forcesPA, rw public_announce_var, intro h1, exact h1,
intro h1, rw public_announce_var, exact h1},
{split, rw forcesPA, rw public_announce_neg, intro h1, exact h1,
intro h1, rw public_announce_neg, exact h1},
{split, rw forcesPA, rw public_announce_conj, intro h1, exact h1,
intro h1, rw public_announce_conj, exact h1},
{split, rw forcesPA, rw public_announce_imp, intro h1, exact h1,
intro h1, rw public_announce_imp, exact h1},
{split, rw forcesPA, rw public_announce_know, intro h1, exact h1,
intro h1, rw public_announce_know, exact h1},
{split, intro h1, rw compositionPA.public_announce_comp,
exact h1, intro h1, rw compositionPA.public_announce_comp at h1, exact h1}
end
```

### C.0.3 Completeness for System **S5** (Epistemic Logic)

```
theorem completenessS5 (hax : sem_cons (∅ : ctx agents) equiv_class)
  (φ : form agents) : global_sem_csq ∅ equiv_class φ → prfS5 ∅ φ :=
begin
rw ←not_imp_not, intro h1,
have h2 := comphelper φ hax h1,
have h3 := lindenbaum {¬φ} h2,
simp at *,
cases h3 with Γ' h3, cases h3 with h3 h4,
rw global_sem_csq,
push_neg,
let f := canonical,
use f,
let v := val_canonical,
split,
exact S5_equiv hax,
use v,
let xΓ' : f.states := ⟨Γ', h3⟩,
split,
exact forcesAX hax,
use xΓ',
have h5 : forces f v xΓ' (¬φ) ↔ ((¬φ) ∈ xΓ'.val),
  from truth hax xΓ' ¬φ,
cases h5 with h5 h6,
have h7 : ¬forces f v xΓ' φ ↔ forces f v xΓ' ¬φ,
  from not_forces_imp f v xΓ' φ,
cases h7 with h7 h8, apply h8, apply h6, exact h4
end
```

### C.0.4 Completeness for System **PA** (Public Announcement Logic)

```
theorem equiv_translation_aux' {Γ : ctxPA agents} (n : nat) (φ : formPA agents)
(h : complexity φ ≤ n) : prfPA Γ (φ ↔ to_PA (translate φ)) :=
begin
  simp at *,
  induction n with n ih generalizing φ,
  { have h1 : complexity φ > 0, from comp_gt_zero, linarith},
  cases φ,
  case formPA.bot
    { exact mp (mp pl4 iden) iden },
  case formPA.var : m
    { exact mp (mp pl4 iden) iden },
  case formPA.and : φ ψ
    { rw translate,
      exact iff_iff_and_iff (ih φ (compand1 h)) (ih ϕ (compand2 h))
    },
  case formPA.impl : φ ψ
    { rw translate,
      repeat {rw to_PA},
      exact iff_iff_imp_iff (ih φ (compimp1 h)) (ih ψ (compimp2 h))
    },
  case formPA.box : a φ
    { simp at *,
      have h1 : complexity φ ≤ n,
      from nat.lt_succ_iff.mp (nat.one_add_le_iff.mp h),
      exact iff_k_dist (ih φ h1),
    },
  case formPA.update : φ ψ
    { cases ψ,
      case formPA.bot
        { repeat {rw translate},
          repeat {rw complexity at h},
          have h1 : complexity φ ≤ n,
            {have : complexity φ + 1 ≤ nat.succ n, linarith,
             exact nat.lt_succ_iff.mp this},
          have h2 := atomicbot,
          exact update_iff1 (ih φ h1) h2
        },
      case formPA.var : m
        { repeat {rw translate},
          repeat {rw complexity at h},
          have h1 : complexity φ ≤ n,
            {have : complexity φ + 1 ≤ nat.succ n, linarith,
             exact nat.lt_succ_iff.mp this},
```

```
        have h2 := atomicperm,
        exact update_iff1 (ih φ h1) h2
      },
    case formPA.and : ψ χ
      { repeat {rw translate},
        exact update_iff2 (ih (U φ ψ) (updatecompand1 h)) (ih (U φ χ)
        (updatecompand2 h)) announceconj,
      },
    case formPA.impl : ψ χ
      { repeat {rw translate},
        exact update_iff3 (ih (U φ ψ) (updatecompimp1 h)) (ih (U φ χ)
        (updatecompimp2 h)) announceimp,
      },
    case formPA.box : a ψ
      { repeat {rw translate},
        exact update_iff4 announceknow (ih (φ ⊇ K a (U φ ψ))
        (updatecompknow2 h))
      },
    case formPA.update : ψ χ
      { rw translate,
        exact update_iff5 announcecomp (ih (U (φ & (U φ ψ)) χ)
        (updatecompupdate h))
      }
  }
end

theorem equiv_translation (Γ : ctxPA agents) :
  ∀ φ : formPA agents, prfPA Γ (φ ↔ to_PA (translate φ)) :=
begin
intro φ,
have h : complexity φ ≤ complexity φ + 1, linarith,
simp,
exact equiv_translation_aux (complexity φ + 1) φ h
end
```

```
theorem completenessPA {φ : formPA agents} (Γ : ctxPA agents) :
  global_sem_csqPA ∅ equiv_class φ → prfPA ∅ φ :=
begin
intros h1,
have h2 := mp pl5 (equiv_translation ∅ φ),
have h3 := soundnessPA,
have h4 : global_sem_csqPA ∅ equiv_class (to_PA (translate φ)),
{intros f h4 v h5 x, exact h3 h2 f h4 v h5 x (h1 f h4 v h5 x)},
have h5 := (global_sem_csqPA_iff_global_sem_csq equiv_class (translate φ) ).mp h4,
have h6 := canonical.completenessS5 sem_consS5 (translate φ),
have h7 := to_prfPA (h6 h5),
simp at *, exact mp (mp pl6 (equiv_translation ∅ φ)) h7
end
```