UNIVERSITATEA DIN BUCUREȘTI

FACULTATEA DE MATEMATICĂ ȘI INFORMATICĂ

SPECIALIZAREA INFORMATICĂ

Lucrare de licență

A FORMALIZATION OF HYBRID LOGIC IN LEAN

Absolvent
Andrei-Alexandru Oltean

Coordonator științific
Prof. dr. Laurențiu Leuștean

București, septembrie 2023

Contents

Chapter 1

Introduction

Our aims with the present thesis are twofold. First, we intend to offer a detailed presentation of a system of modal logic known as *hybrid logic*. Our purpose is to walk the reader through all relevant aspects of this logic, including its syntax and semantics, and culminating with two important properties known as *soundness* and *completeness*. Second, we mean to illustrate the way in which computers can assist mathematicians in the business of proving formal statements. In this respect, we will turn to the Lean 4 proof assistant; a project started in 2013 by Leonardo de Moura, which has since allowed for the expression of large areas of mathematics in computer language. As a case study into the capabilities of Lean, we will attempt to translate the logic defined here and its formal properties into statements of this programming language. A side effect of our work, we hope, is that it may also provide indications about the way in which lines of reasoning can have direct analogues in lines of code.

Hybrid logic originated in the philosophical works of Arthur Prior [35]. His contribution came in light of an influential analysis by John McTaggart [32], who had identified two opposing conceptions of time. The first, McTaggart referred to as the A-series theory of time. It is the view that events are ordered in time through the ever-changing property of being either in the past, present or future. It roughly corresponds to time as it is experienced from a first-person level, locally. The B-series theory, on the other hand, claims an external perspective on time. According to it, events happen at determined temporal instances, and all instances are related to each other in a static, determined order. Prior's belief was that the dispute could be settled through logical analysis. He thus turned to various modal logics as a means of studying the true nature of time. To formalize A-series talk, he invented tense logic. It is a type of modal logic equipped with two modal operators: one intended to state properties about the future, and the other reserved for statements about the past. Hybrid logic was Prior's response to the B-series theory. For though he was determined to show that tense logic was sufficient to capture the essence of temporal discourse, the best suited formalism for the B-series conception seemed to be that of first-order logic [16], interpreted over a domain of temporal instances. According to [4], Prior was weary of what he saw as a dubious ontological commitment in this first-order approach. His solution was to enrich the vocabulary of tense logic with new symbols, by which instances of time could be directly referenced, without committing to their existence as entities. These symbols are now known as *nominals*. Their addition is what sets hybrid logics apart from regular modal logic.

Yet hybrid logics, and modal logics in general, are far from being tools of interest only within the confines of metaphysical investigation. Blackburn et. al [6] express this fact succinctly thus: "Modal languages are not isolated formal systems". Thanks to the development of Kripke semantics, modal languages prove to be a powerful formalism for studying relational structures in general. For example, various modal languages have been proposed as database query languages [8, 13, 7, 10], with hybrid languages a popular choice for modelling XML constraints and queries [3, 11, 12]. Programs themselves can be regarded as examples of relational structures, by modelling the dynamics of their execution under all possible inputs as a directed graph. This makes modal logics well-suited for applications to formal verification, where we wish to rigorously check that programs satisfy various correctness constraints (i.e., that they are bug-free). Propositional Dynamic Logic [18] has been a standard way to express program specifications. More recent developments include Matching Logic [36], which forms the theoretical basis of the K Framework [37]. In this respect, hybrid languages prove once again to be an interesting area of study, since multisorted polyadic hybrid logic has been shown to be equivalent to Matching Logic [22].

Here, we deal with the usual, monosorted variety of hybrid logic. We offer a Lean formalization of its proof system and semantics, we prove some theorems within this system, and some metatheorems like Soundness and Deduction. We also establish a clear path towards a full formalization of Completeness, the proof of which we have only partially translated to Lean. Some other modal logics have been formalized in Lean in the past. Of these, we mention S5 [2], Public Announcement Logic [23] and Matching Logic [9]. However, we have not identified any previous formalization of a system of hybrid logic in a proof assistant. Our formalization is available at [34].

We begin our investigation of hybrid logic by studying its properties in natural mathematical language. In Chapter 2, we define the syntax of hybrid logic, we describe its semantics, and we prove some hybrid theorems in a Hilbert-style proof system. In Chapter 3, we detail the first result linking the semantics of hybrid logic to its proof system: Soundness. In Chapter 4, we go on to prove Completeness, after providing a thorough proof of Lindenbaum's Lemma. The general method we employ follows that offered in [5]. Chapter 5 presents the Lean proof assistant and our main design choices in formalizing hybrid logic. Finally, in Chapter 6 we lay down the conclusions of our work and provide directions for further research.

Chapter 2

Hybrid Languages: Syntax and Semantics

In this section, we define the syntax of hybrid logic, we describe its semantics, and we prove some hybrid theorems in a Hilbert-style proof system which we introduce. Since hybrid languages are extensions of modal languages, both syntactically and semantically they build upon standard modal conventions. We use [6] and [15] as a reference for the definition of the language and its Kripke semantics. For the characteristically hybrid part of this section we referred to [5]. Our treatment of substitutions and the Re-Replacement Lemma follows the indications given in [14].

2.1 Basic Syntax. Substitutions

At its core, a language is determined by a collection of symbols, which can be variously composed. We call this collection of basic symbols the *signature* of the language. In hybrid languages, the symbols in the signature can be of three sorts: propositional symbols, nominal symbols and variable symbols. We define this idea precisely below.

Definition 2.1.1 (Signatures)**.** A **hybrid signature** $\mathscr{L}$ is a tuple $\langle \text{PROP}, \text{SVAR}, \text{NOM} \rangle$, where:

1. $\text{PROP} = \{p, q, r, ...\}$ is a denumerably infinite set

2. $\text{SVAR} = \{x, y, z, ...\}$ is a denumerably infinite set

3. NOM $= \{i, j, k, ...\}$ is a finite or denumerably infinite set

By "atomic symbol" we will mean any member of the set ATOM $=$ PROP $\cup$ SVAR $\cup$ NOM. Of all atomic symbols, NOM and SVAR taken together make up the set of *state symbols*. The composition of atomic symbols renders *formulas*, which give rise to genuine languages:

Definition 2.1.2 (Well-Formed Formulas)**.** Given a signature $\mathscr{L}$, we define the notion of a $\mathscr{L}(\forall)$ **-well-formed formula (**$\mathscr{L}(\forall)$ **-WFF)** thus:

$$\mathscr{L}(\forall)\ \textbf{-WFF} := \bot \mid a \mid \varphi \rightarrow \psi \mid \Box\varphi \mid \forall x\ \varphi,$$

for any $a \in$ ATOM and $x \in$ SVAR.

We will denote the set of $\mathscr{L}(\forall)$ -well-formed formulas as $Form_{\mathscr{L}(\forall)}$. For brevity, we may refer to $\mathscr{L}(\forall)$ -well-formed formulas simply as $\mathscr{L}(\forall)$ -*formulas*. We will employ the following abbreviations for boolean and modal operators, and for the existential quantifier:

(i) $\neg\varphi := \varphi \rightarrow \bot$

(ii) $\varphi \wedge \psi := \neg(\varphi \rightarrow \neg\psi)$

(iii) $\varphi \leftrightarrow \psi := (\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$

(iv) $\varphi \vee \psi := \neg\varphi \rightarrow \psi$

(v) $\Diamond\varphi := \neg\Box\neg\varphi$

(vi) $\exists x\ \varphi := \neg\forall x\,\neg\varphi$

As in first-order logic, notice that we can bind variable symbols to quantifiers. Furthermore, as in propositional modal logic, we dispose of the modal operators which we can apply to formulas. Thus we can already see one reason why such languages are called "hybrid": syntactically speaking, they are a hybrid between propositional modal logic and standard first-order logic. Unlike first-order systems, though, notice that state variables *also function as propositional symbols*. That is, x is both a symbol that can be bound, in a formula like $\forall x\ \varphi$; but x is also a well-formed formula by itself. That is another way such languages can be thought of as *hybrid*: symbols in SVAR are a hybrid between variables and propositions. Why would such a hybridization of variables be of interest? The intention is to make state symbols function as *labels*. This notion should become clear once we provide the semantics for such languages.

In the preceding paragraph we mentioned *bound variables*. Here is an inductive definition to show precisely what we mean by that:

Definition 2.1.3 (Free Variables)**.** We will say a variable x is **free** in a $\mathscr{L}(\forall)$ -formula φ if and only if:

(i) if $\varphi = \bot$ or $\varphi \in$ ATOM, then x is free in φ iff $\varphi = x$

(ii) if $\varphi = \Box\,\psi$, then x is free in φ iff x is free in ψ

(iii) if $\varphi = \psi \to \chi$, then x is free in φ iff x is free in ψ or x is free in χ

(iv) if $\varphi = \forall y\ \psi$, then x is free in φ iff $x \neq y$ and x is free in ψ

By the above definition, if x is *not free* in φ, then either x does not occur at all in φ, or x occurs in some subformula $\forall x\ \psi$ of φ. If a variable *x occurs in φ* and *is not free in φ*, we will say x is **bound** in φ.

Our intention will be to ensure that free and bound occurrences of the same variable bear only a *graphic* similarity to one another. On a semantic and intuitive level, *they should mean completely different things*. Take the formula $\Diamond x \to \forall x\ \Diamond x$. The free occurrence of x acts much like a constant, in that it designates a *given, fixed entity*; and we may dispose of a theory which makes *additional assumptions* about *the* specific entity designated by x. The bound occurrence of x that succeeds it, however, shares none of the assumptions which we might make about x. Instead of referring to a given entity, a bound occurrence of a variable acts more like a *varying name*; it refers to *any entity at all, arbitrarily chosen*. Only when the free occurrence of x is truly arbitrary (i.e., if we make *no* additional assumptions about it) will we be entitled to treat it the same as a universally quantified variable. (In the case of hybrid logic, the entities we refer to are states in a graph. We will return to that point when we define the semantics.)

Now we can move to substitutions:

Definition 2.1.4 (Substitutions)**.** The $\mathscr{L}(\forall)$ -formula $\varphi\ [y/x]$, obtained by substituting in φ each free occurrence of x with an occurrence of y, is defined thus:

(i) if $\varphi = \bot$, $\varphi \in \text{PROP}$ or $\varphi \in \text{NOM}$, then $\varphi\ [y/x] = \varphi$

(ii) if $\varphi \in \text{SVAR}$, then $\varphi\ [y/x] = y$ if $\varphi = x$, and $\varphi\ [y/x] = \varphi$ otherwise

(iii) if $\varphi = \psi \to \chi$, then $\varphi\ [y/x] = \psi\ [y/x] \to \chi[y/x]$

(iv) if $\varphi = \Box\, \psi$, then $\varphi\ [y/x] = \Box\, \psi\ [y/x]$

(v) if $\varphi = \forall z\ \psi$, then $\varphi\ [y/x] = \forall z\ \psi\ [y/x]$ if $z \neq x$, and $\varphi\ [y/x] = \varphi$ otherwise

We claim the operation defined above replaces all free occurrences of x in φ. Let us check.

Proposition 2.1.1. Let φ be a $\mathscr{L}(\forall)$ -formula and x, y two *distinct* variables in $\mathscr{L}$. Then x does not occur free in $\varphi\ [y/x]$.

Proof. We will proceed by induction on formulas.

If $\varphi \in \bot$, $\varphi \in \text{PROP}$ or $\varphi \in \text{NOM}$, then no substitution at all occurs, so $\varphi\ [y/x] = \varphi$. By assumption no variable occurs in φ, so x does not occur free in φ.

Next, the case when $\varphi = z$, for some state variable z. If $x \neq z$, then once again no substitution occurs, meaning $\varphi\ [y/x] = \varphi = z$. If $x = z$, then $\varphi = x$, and by applying the definition above we get $\varphi\ [y/x] = y$. In both cases $\varphi\ [y/x]$ is equal to some state variable distinct from x, so by applying the definition of free occurrences we conclude that x is not free in $\varphi\ [y/x]$.

Now let $\varphi = \psi \to \chi$, and assume x does not occur free in either ψ or χ. Applying the definition of substitutions, we know that $\varphi\ [y/x] = (\psi \to \chi)\ [y/x] = \psi\ [y/x] \to \chi\ [y/x]$. Unfolding the definition, x is not free in this formula iff x is not free in $\psi\ [y/x]$ and x is not free in $\chi\ [y/x]$. This is provided to us by the induction hypothesis.

We proceed similarly when $\varphi = \Box\, \psi$. Assume x does not occur free in ψ. Now, x does not occur free in $\varphi\ [y/x] = (\Box\, \psi)\ [y/x]$ iff x does not occur free in $\Box(\psi\ [y/x])$, iff x does not occur free in $\psi\ [y/x]$. By the induction hypothesis, this is true.

Finally, assume $\varphi = \forall z\ \psi$ and x does not occur free in $\psi\ [y/x]$. We once again distinguish the case that $x = z$ from the case that $x \neq z$. If $x = z$, then $\varphi\ [y/x] = \varphi = \forall x\ \psi$. Clearly x is not free in $\forall x\ \psi$. Now, if $x \neq z$, then $\varphi\ [y/x] = \forall z\ (\psi\ [y/x])$, and x does not occur free in $\forall z\ (\psi\ [y/x])$ iff it does not occur free in $\psi\ [y/x]$. We apply the induction hypothesis and reach the desired conclusion.

$\square$

So substitution works as expected, by getting rid of all free occurrences of the variable we apply it to. For an example, take the formula $\varphi = y \rightarrow \forall y\ (x \rightarrow \Box \neg y)$. The first occurrence of y and the first occurrence of x are free; the last occurrence of y is bound. By replacing all free occurrences of y by x, we get $\varphi\ [x/y] = x \rightarrow \forall y\ (x \rightarrow \Box \neg y)$. And by replacing all free occurrences of x with y, we get $\varphi\ [y/x] = y \rightarrow \forall y\ (y \rightarrow \Box \neg y)$.

In practice, however, the substitution $\varphi\ [y/x]$ is one we will never want to make. To see why, take a look at φ: notice that x occurs free under the scope of y. Now, if we look at $\varphi\ [y/x]$, we see that the free occurrence of x has been replaced by an occurrence of y. *But the new occurrence of y is no longer free*, because x was in the scope of a y-quantifier in the original formula; thus in the substituted formula y is bound by that quantifier. So we transformed a free occurrence of a variable into a bound occurrence, fundamentally changing the meaning of the formula.

We must then introduce a means of avoiding such problematic substitutions. Consider this definition.

Definition 2.1.5 (Substitutability)**.** Given a $\mathscr{L}(\forall)$ -formula φ and two variables x and y (not necessarily distinct), we will say y **is substitutable for** x **in** φ:

(i) if $\varphi = \bot$ or $\varphi \in$ ATOM;

(ii) if $\varphi = \psi \rightarrow \chi$, then y is substitutable for x in φ if y is substitutable for x in ψ and y is substitutable for x in χ;

(iii) if $\varphi = \Box\, \psi$, then y is substitutable for x in φ if y is substitutable for x in ψ;

(iv) if $\varphi = \forall z\ \psi$, then y is substitutable for x in φ if x is not free in ψ, or $z \neq y$ and y is substitutable for x in ψ.

We saw that substituting y for x when x occurs free under the scope of y has unintended consequences. The definition we gave for substitutability guards precisely against this situation. So in the formula φ we considered earlier, $\varphi = y \rightarrow \forall y\ (x \rightarrow \Box \neg y)$, y is *not* substitutable for x. We can still perform the substitution $\varphi\ [y/x]$ and obtain a new formula, but the failure of substitutability should warn us that whatever reasoning we make about $\varphi\ [y/x]$ may *not* carry over to φ.

Here is another fact about substitutions which we are going to need:

Proposition 2.1.2. Let φ be a $\mathscr{L}(\forall)$ -formula and x, y two variables in $\mathscr{L}$. If x does not occur free in φ, then $\varphi\,[y/x] = \varphi$. Additionally, if x does not occur at all in φ, then y is substitutable for x in φ.

Proof. If $\varphi = \bot$, $\varphi \in$ NOM or $\varphi \in$ PROP, then by definition $\varphi\,[y/x] = \varphi$; and also by definition y is substitutable for x in φ. So both claims are true.

Suppose now $\varphi \in$ SVAR. Clearly, $\varphi \neq x$, since otherwise x would occur free in φ. So $\varphi = z$, for some $z \neq x$. For the first part of our claim, notice $z\,[y/x] = z$, so $\varphi\,[y/x] = \varphi$, as requested. Next, φ is an atomic formula, so y is substitutable for x in φ. This proves the second part the proposition.

Moving to the inductive cases, let us have $\varphi = \psi \to \chi$ and assume the proposition holds for ψ and χ. If x does not occur free in $\psi \to \chi$, then it does not occur free in either ψ or χ. We learn from the induction hypothesis, then, that $\psi\,[y/x] = \psi$ and $\chi\,[y/x] = \chi$. So we obtain $\varphi\,[y/x] = (\psi \to \chi)\,[y/x] = (\psi \to \chi) = \varphi$.

Now we want to show that if x does not occur in φ, then y is substitutable for x in φ. In a similar fashion as before, if x does not occur at all in $\psi \to \chi$, then x does not occur in either ψ or χ. Applying the induction hypothesis we find that y is substitutable for x in both ψ and χ; so y is substitutable for x in $(\psi \to \chi) = \varphi$.

Consider now the modal case: suppose $\varphi = \Box\,\psi$ and x does not occur free in $\Box\,\psi$. This means x does not occur free in ψ, so by the induction hypothesis $\psi\,[y/x] = \psi$. By definition, $\varphi\,[y/x] = \Box(\psi\,[y/x])$. So $\varphi\,[y/x] = \Box\,\psi = \varphi$. Next, assume x does not occur in $\varphi = \Box\,\psi$, meaning x does not occur in ψ. We apply the induction hypothesis to conclude that y is substitutable for x in ψ; so y is substitutable for x in $\Box\,\psi = \varphi$.

For the bound case, $\varphi = \forall z\,\psi$. If x does not occur free in φ, then by definition y is substitutable for x in φ, so we only need to show $\varphi\,[y/x] = \varphi$. Suppose first that $\varphi = \forall x\,\psi$. In this case it follows immediately that $\varphi\,[y/x] = \varphi$. Finally, if $\varphi = \forall z\,\psi$ for some $z \neq x$, then $\varphi\,[y/x] = \forall z\,(\psi\,[y/x])$. Since x does not occur free in $\forall z\,\psi$ and $z \neq x$, it follows that x does not occur free in ψ. We can thus apply the induction hypothesis, obtaining $\psi\,[y/x] = \psi$, so $\varphi\,[y/x] = \forall z\,\psi = \varphi$. $\square$

The reason this last result is useful is because it simplifies the next proof we want to make. We want to show that, *under some conditions*, substitutions can be *reversed*. The following lemma gives the required restrictions.

Lemma 2.1.3 (Re-Replacement Lemma)**.** Let φ be a $\mathscr{L}(\forall)$ -formula and x, y two variables. If y does not occur in φ and y is substitutable for x in φ, then x is substitutable for y in $\varphi\ [y/x]$ and $\varphi\ [y/x][x/y] = \varphi$.

Proof. If $\varphi = \bot$, $\varphi \in$ NOM or $\varphi \in$ PROP, there are no variables to substitute, meaning $\varphi\ [y/x][x/y]$ is definitionally equal to $\varphi\ [x/y]$, which is in turn equal to φ. Similarly, we conclude x is substitutable for y in $\varphi\ [y/x]$.

If $\varphi = z$, for $z \in SVAR$, then $\varphi\ [y/x] = z\ [y/x]$ is equal to either y or z, according to whether $z = x$ or $z \neq x$. In both cases, x is substitutable for y in $\varphi\ [y/x]$, since it is an atomic formula. Moreover, if $x = z$, then $\varphi\ [y/x][x/y] = y\ [y/x] = x = z$, which is in turn equal to φ. If $x \neq z$, then $\varphi\ [y/x][x/y] = z\ [x/y]$. By the hypothesis that y does not occur in φ, we know $y \neq z$, so $z\ [x/y] = z = \varphi$.

Let us now have $\varphi = \psi \to \chi$ and assume by induction that the property holds for ψ and χ. To show that x is substitutable for y in $(\psi \to \chi)\ [y/x]$, we need to show it is substitutable in $\psi\ [y/x]$ and in $\chi\ [y/x]$. And since $(\psi \to \chi)\ [y/x][x/y] = \psi\ [y/x][x/y] \to \chi\ [y/x][x/y]$, we need to show that $\psi\ [y/x][x/y] = \psi$ and $\chi\ [y/x][x/y] = \chi$. Since y does not occur free in $\psi \to \chi$, it follows that it does not occur free in either of the two, we can apply the induction hypotheses to ψ and χ and arrive at the result.

For $\varphi = \Box\, \psi$ we behave similarly to the boolean case, but now we only have to treat with one induction hypothesis, namely for ψ. The proof steps remain identical.

For the bound case, $\varphi = \forall z\ \psi$, consider first $z = y$. In this case, we know by hypothesis that y is substitutable for x in $\forall y\ \psi$. This can only be true if x is not free in ψ. Clearly then x is not free in $\forall y\ \psi$ either. So we apply Proposition 2.1.2 and find that $(\forall y\ \psi)\ [y/x] = \forall y\ \psi$. Our goal then becomes to show that x is substitutable for y in $\forall y\ \psi$ and that $(\forall y\ \psi)\ [x/y] = \forall y\ \psi$. By hypothesis, we know y does not occur at all in $\forall y\ \psi$. So by applying Proposition 2.1.2 once again, this time to y and $\forall y\ \psi$, we immediately prove our goal.

Moving on to the case that $z \neq y$. If $z \neq x$, then, $(\forall z\ \psi)\ [y/x] = \forall z\ (\psi\ [y/x])$. So we need to show that x is substitutable for y in $\forall z\ (\psi\ [y/x])$ and that $\psi\ [y/x][x/y] = \varphi$. To reach this, we can apply the induction hypothesis, granted we know that y does not occur in ψ (which we do, since y does not occur in $\forall \psi$), and that y is substitutable for x in ψ. So is it? We *do* know that y is substitutable for x in $\forall z\ \psi$, but from this we can only reach the desired conclusion *if x is free in ψ*. If x is *not* free in ψ, the hypothesis that y is substitutable for x in $\forall z\ \psi$ is simply true by definition, so it tells us nothing valuable. How can we proceed? Just as we did before, we apply Proposition 2.1.2 twice. First we apply it to x and y in ψ, and obtain $\psi\ [y/x] = \psi$; then we apply it to y and x in $\psi\ [y/x] = \psi$, and obtain $\psi\ [y/x][x/y] = \psi$ and x is substitutable for y in $\forall z\ (\varphi\ [y/x])$, as required.

Finally, we need to treat the case $x \neq y$ and $z = x$. So $\varphi = \forall x\ \psi$. Clearly, x does not occur free in φ, so by Proposition 2.1.2 $\varphi\ [y/x] = \varphi$. Then, since y does not occur at all in φ, we arrive at $\varphi\ [y/x][x/y] = \varphi$ and x is substitutable for for y in $\varphi\ [y/x]$.

□

Finally, we introduce a mechanism of substituting bound variables:

Definition 2.1.6 (Free Variants)**.** Let $v_0, v_1, \ldots$ be an enumeration of all $\mathscr{L}(\forall)$ -variables. For any $\mathscr{L}(\forall)$ -formula φ and any $\mathscr{L}(\forall)$ -variable x, we define φ', the *x-free variant of* φ, thus:

(i) if $\varphi = \forall x\ \psi$, then $\varphi' = \forall v\ \psi'\ [v/x]$, where v is the first variable in $v_0, v_1, \ldots$ which is not x and does not occur in ψ'

(ii) if $\varphi = \psi \rightarrow \chi$, then $\varphi' = \psi' \rightarrow \chi'$

(iii) if $\varphi = \Box\, \psi$, then $\varphi' = \Box\, \psi'$

(iv) in all other cases, $\varphi' = \varphi$

2.2 Semantics

To interpret $\mathscr{L}(\forall)$ -formulas, we'll borrow the idea of assignment functions from first-order logics and apply it to Kripke models. Let us begin with the latter.

Definition 2.2.1 (Kripke Models)**.** A **Kripke model** $\mathscr{M}$ for a language $\mathscr{L}(\forall)$ over PROP, SVAR and NOM is a tuple $\langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$, where W is a non-empty set, R is a binary relation on W, and V is a function $\mathrm{V}\colon\ \mathrm{PROP} \cup \mathrm{NOM} \rightarrow \mathscr{P}(\mathrm{W})$.

We will use the terms "Kripke model" and "model" interchangeably. The tuple $\langle \mathrm{W}, \mathrm{R} \rangle$ defines the *Kripke frame* of the model. Any Kripke frame creates a directed graph structure: the set W comprises its nodes, while the relation R comprises its edges. We call W the set of *states* of the frame (alternatively, the set of *possible worlds*, which justifies using the letter W). We call R the *accessibility relation* of the frame.

So what do directed graphs have to do with logics and languages? Intuitively, propositional modal logic arises when we map propositional truth-assignments (i.e., functions that assign either T or F to every propositional symbol) to *nodes* in a graph. One way of viewing Kripke models is that every state in its frame has *its own* truth-assignment. This allows us to reason about the truth of propositions under various *traversals* of the graph: is φ true in all nodes accessible from the current node? The job of the modal operators is to answer such questions.

We mentioned that every state in a Kripke model has its own truth-assignment. Equivalently, we could have said that every propositional symbol is assigned to be true at *some* of the states. That is precisely the role of the function V, the **valuation function**: it maps symbols to sets of states.

Now, on top of the basic modal language, in hybrid languages we have two more sorts of atomic symbols to take care of: we have nominals and variables. By extending the modal language this way, we intend to also have the means to *label* the states in the graph, and to *quantify* over them. What we mean by "labeling" is that, ideally, nominals and variables should be evaluated as true at just *one* unique state.

Let's begin with nominals first. In accordance with our desire that nominals should behave as labels, we will concentrate on valuations V which satisfy this property: if $i \in \mathrm{NOM}$, then $\mathrm{V}(i) = \{s\}$, a singleton set of states. Such a valuation we will call a **standard valuation**. Given a model $\mathscr{M} = \langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$, if V is standard, then we will accordingly say $\mathscr{M}$ is a **standard model**.

Now on to state variables. Since we want to be able to bind them, their interpretation will be given separately from the valuation V, namely by assignment functions:

Definition 2.2.2 (Assignment Functions)**.** Given $\mathscr{M} = \langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$, any function $\mathrm{g} \colon \mathrm{SVAR} \to \mathscr{P}(W)$, which maps state variables to sets of elements in W, is called an **assignment function**.

If for all $x \in \mathrm{SVAR}$, $\mathrm{g}(x)$ is a singleton set, then g is a **standard assignment**.

Definition 2.2.3 (Assignment Variants)**.** Given two assignment functions g and g' and a state variable $x \in SVAR$, we say g' is an **x-variant** of g' if and only if, for all $y \in SVAR$, if $y \neq x$ then $\mathrm{g}'(y) = \mathrm{g}(x)$. We will denote this relation by $\mathrm{g}' \overset{\mathrm{x}}{\leftrightsquigarrow} \mathrm{g}$.

Now we have all the ingredients needed to define the satisfaction relation for $\mathscr{L}(\forall)$.

Definition 2.2.4 (Satisfaction)**.** Let $\mathscr{M} = \langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$ be a standard model, g a standard assignment and $\mathrm{s} \in \mathrm{W}$. The **satisfaction relation** for a language $\mathscr{L}(\forall)$, $\models$, is defined as follows:

$$\begin{aligned}
\mathscr{M}, \mathrm{s}, \mathrm{g} \models \bot &\iff \textbf{False} \\
\mathscr{M}, \mathrm{s}, \mathrm{g} \models a &\iff s \in \mathrm{V}(a), \text{ where } a \in \mathrm{PROP} \cup \mathrm{NOM} \\
\mathscr{M}, \mathrm{s}, \mathrm{g} \models x &\iff s \in \mathrm{g}(x), \text{ where } x \in \mathrm{SVAR} \\
\mathscr{M}, \mathrm{s}, \mathrm{g} \models \varphi \to \psi &\iff \mathscr{M}, \mathrm{s}, \mathrm{g} \models \varphi \text{ implies } \mathscr{M}, \mathrm{s}, \mathrm{g} \models \psi \\
\mathscr{M}, \mathrm{s}, \mathrm{g} \models \Box \varphi &\iff \text{ for all } s' \in W, \text{ if } sRs' \text{ then } \mathscr{M}, \mathrm{s}', \mathrm{g} \models \varphi \\
\mathscr{M}, \mathrm{s}, \mathrm{g} \models \forall x\ \varphi &\iff \text{ for all assignment functions } \mathrm{g}', \text{ if } \mathrm{g}' \overset{\mathrm{x}}{\leftrightsquigarrow} \mathrm{g} \text{ then } \mathscr{M}, \mathrm{s}, \mathrm{g}' \models \varphi
\end{aligned}$$

When the context might provoke ambiguity about the language in question, we may write $\models_{\mathscr{L}(\forall)}$ instead of $\models$.

If $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \varphi$, then we say φ is *satisfied* in $\mathscr{M}$ at s under g. A formula φ is *satisfiable* if there exists a standard model $\mathscr{M}$, some state s and a standard assignment g such that $\mathscr{M}, s, g \models \varphi$.

We say φ is *valid* if for all standard models $\mathscr{M} = \langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$, all states s and all standard assignment functions g, $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \varphi$. We will abbreviate this statement by writing simply $\models \varphi$.

A set of formulas Γ is *satisfied* in $\mathscr{M}$ at s under g if, for all formulas $\varphi \in \Gamma$, $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \varphi$. Clearly, a formula φ is satisfied if and only if the set $\{\varphi\}$ is satisfied. A set of formulas Γ is *satisfiable* if there exist $\mathscr{M}$, s and g such that $\mathscr{M}, s, g \models \Gamma$. We say Γ *entails* φ if, for all $\mathscr{M}$, s and g, $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \Gamma$ implies $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \varphi$. We write this as $\Gamma \models \varphi$.

Notice there is no constraint on the accessibility relation R to be transitive. That is, from the truth of $\Box\varphi$ we will not, in general, be able to infer the truth of $\Box\Box\varphi$. On the contrary, we will often find ourselves reasoning about formulas with multiple, iterated modal operators, as in "$\Box^n\varphi$"; by which we mean the formula obtained by writing the box operator n times, for some $n > 0$. On the semantic level, such formulas speak of *paths* on the frame. Since we will frequently encounter them, it helps to clarify these notions now.

Definition 2.2.5 (k-step transitions)**.** For any model $\langle \mathrm{W}, \mathrm{R}, \mathrm{V}\rangle$ and $w, w' \in \mathrm{W}$, we will say there is a *k-step transition* between w and w' if, for some $k > 0$, there exist $s_0, ..., s_k \in \mathrm{W}$ so that $s_0 = w, s_k = w'$ and $s_i \mathrm{R} s_{i+1}$ for all $0 \leq i < k$.

Proposition 2.2.1. Let $\mathscr{M} = \langle \mathrm{W}, \mathrm{R}, \mathrm{V}\rangle$ be a model, s $\in$ W and g and assignment function. Then, for all $n > 0$:

(i) $\mathscr{M}, s, g \models \Box^n\varphi$ iff $\mathscr{M}, s', g \models \varphi$, for all s$'$ $\in$ W such that there is a n-step transition between s and s';

(ii) $\mathscr{M}, s, g \models \Diamond^n\varphi$ iff $\mathscr{M}, s', g \models \varphi$, for some s$'$ $\in$ W such that there is a n-step transition between s and s'.

Proof. For $n = 1$, this follows by definition. For the inductive step, notice that if there is an $m+1$-step transition between s and s$'$, then there is some state s$''$ such that there is an m-step transition between s and s$''$, and s$''$ R s. Applying the induction hypothesis to s and s$''$ leads to the desired conclusion. $\Box$

Before we move on to the proof system, let us prove a small fact we'll need later: to be satisfiable is the same thing as not being contradictory.

Proposition 2.2.2. *Let Γ be a set of formulas. Γ is satisfiable if and only if $\Gamma \not\models \bot$.*

Proof. By unfolding the definitions, we obtain:

$$\begin{aligned} \Gamma \not\models \bot &\iff \text{there are } \mathscr{M}, s, g \text{ such that } \mathscr{M}, s, g \models \Gamma \text{ and } \mathscr{M}, s, g \not\models \bot \\ &\iff \text{there are } \mathscr{M}, s, g \text{ such that } \mathscr{M}, s, g \models \Gamma \\ &\iff \Gamma \text{ is satisfiable} \end{aligned}$$

$\Box$

Proposition 2.2.3. *Let Γ be a set of formulas. Γ is unsatisfiable if and only if $\Gamma \models \bot$.*

Proof. By contraposition from Proposition 2.2.2. $\square$

2.3 The Proof System

We begin to give the details for a Hilbert-style proof system for hybrid logic. The first step is to define the propositional part of the logic. We thus introduce the familiar notions of propositional evaluation and tautology.

Definition 2.3.1 (Propositional Evaluations)**.** An $\mathscr{L}(\forall)$ -evaluation is a function $e\colon \mathit{Form}_{\mathscr{L}(\forall)} \to \{0,1\}$ which, for all $\mathscr{L}(\forall)$ -formulas φ and ψ, satisfies the following two conditions:

1. $e(\bot) = 0$

2. $e(\varphi \to \psi) = 1$ iff $e(\varphi) = 1$ implies $e(\psi) = 1$

Definition 2.3.2 (Tautologies)**.** An $\mathscr{L}(\forall)$ -formula φ is a tautology if $e(\varphi) = 1$ under any evaluation e.

We list below a list of all propositional tautologies which will be useful in proofs. Since our chief interest lies in the modal and hybrid parts of our language, it is beyond the scope of our treatment to show that they are indeed tautologies. However, their proofs are standard, and we refer the reader to [21] for additional details.

Proposition 2.3.1 (Tautology Instances)**.** For all formulas φ, ψ, χ:

1. $\varphi \to \varphi$ is a tautology;

2. $(\varphi \to \psi) \to (\neg\psi \to \neg\varphi)$ and $(\neg\psi \to \neg\varphi) \to (\varphi \to \psi)$ are tautologies;

3. $\bot \to \varphi$ is a tautology;

4. $\neg\neg\varphi \to \varphi$ and $\varphi \to \neg\neg\varphi$ are tautologies;

5. $\varphi \to \psi \to (\varphi \wedge \psi)$ is a tautology;

6. $(\varphi \wedge \psi) \to \varphi$ and $(\varphi \wedge \psi) \to \psi$ are tautologies;

7. $(\varphi \to \chi) \to ((\varphi \wedge \psi) \to \chi)$ is a tautology;

8. $(\varphi \to (\psi \to \chi)) \to (\varphi \to \psi) \to (\varphi \to \chi)$ is a tautology;

9. $(\varphi \to \psi) \to (\psi \to \varphi) \to (\varphi \leftrightarrow \psi)$ is a tautology;

10. $\neg(\varphi \to \psi) \to (\varphi \wedge \neg\psi)$ and $(\varphi \wedge \neg\psi) \to \neg(\varphi \to \psi)$ are tautologies;

11. $\neg(\varphi \wedge \psi) \to (\varphi \to \neg\psi)$ and $(\varphi \to \neg\psi) \to \neg(\varphi \wedge \psi)$ are tautologies;

12. $(\varphi \leftrightarrow \psi) \leftrightarrow (\neg\varphi \leftrightarrow \neg\psi)$ is a tautology;

13. $((\varphi \wedge \psi) \to \chi) \leftrightarrow (\varphi \to \psi \to \chi)$ is a tautology.

Now we can introduce the logic. For any language $\mathscr{L}(\forall)$, we define $\mathscr{H}_{\mathscr{L}}(\forall)$, **the hybrid logic of** $\mathscr{L}(\forall)$, to be the smallest set of $\mathscr{L}(\forall)$-formulas fulfilling the following conditions:

A. **(Axioms)** For all $\mathscr{L}(\forall)$-formulas φ, ψ, χ, $\mathscr{H}_{\mathscr{L}}(\forall)$ contains an instance of the following formulas:

   1. (T) φ, where φ is a tautology
   2. (K) $\Box(\varphi \to \psi) \to \Box\varphi \to \Box\psi$
   3. (Q1) $\forall x\, (\varphi \to \psi) \to (\varphi \to \forall x\, \psi)$, for all $x \in$ SVAR, if x does not occur free in φ
   4. (Q2) $\forall x\, \varphi \to \varphi\, [a/x]$, for all $a \in \text{SVAR} \cup \text{NOM}$, if a is substitutable for x in φ
   5. (Name) $\exists x\, x$
   6. (Nom) $\forall x\, (\Diamond^n(x \wedge \varphi) \to \Box^m(x \to \varphi))$, for all $n, m \in \mathbb{N}$
   7. (Barcan) $\forall x\, \Box\varphi \to \Box\forall x\, \varphi$

B. **(Rules of inference)** The set $\mathscr{H}_{\mathscr{L}}(\forall)$ is closed under the following rules:

   (i) (Modus Ponens) If $\varphi \to \psi \in \mathscr{H}_{\mathscr{L}}(\forall)$ and $\varphi \in \mathscr{H}_{\mathscr{L}}(\forall)$, then $\psi \in \mathscr{H}_{\mathscr{L}}(\forall)$

   (ii) (Generalization) If $\varphi \in \mathscr{H}_{\mathscr{L}}(\forall)$, then $\forall x\, \varphi \in \mathscr{H}_{\mathscr{L}}(\forall)$ for all $x \in$ SVAR

(iii) (Necessitation) If $\varphi \in \mathscr{H}_{\mathscr{L}}(\forall)$, then $\Box\varphi \in \mathscr{H}_{\mathscr{L}}(\forall)$

For any $\mathscr{L}(\forall)$ -formula φ, if $\varphi \in \mathscr{H}_{\mathscr{L}}(\forall)$ we will write $\vdash_{\mathscr{L}(\forall)} \varphi$ and say φ is a **theorem** of $\mathscr{L}(\forall)$. If the language in question is clear from the context, we will drop the subscripts and write simply $\vdash \varphi$. If Γ is a set of formulas, we will say φ is a **syntactic consequence** of Γ, $\Gamma \vdash \varphi$, if there exists a finite subset $\{\chi_1, ..., \chi_n\}$ of Γ such that $\vdash (\chi_1 \wedge ... \wedge \chi_n) \rightarrow \varphi$. We will say Γ is **consistent** if $\Gamma \nvdash \bot$.

By **"formal derivation of φ"**, we will refer to a finite sequence of formulas $\langle \psi_1, ..., \psi_n \rangle$, such that $\varphi = \psi_n$, and for every i, ψ_i is either an instance of an axiom, or is provable by an application of the rules of inference to some earlier formula(s) in the sequence.

Remember that our end-goal is to show that hybrid logic is both sound and complete. That is, we want to show that there is an equivalence between the semantic approach to the language we introduced in the previous chapter, and the syntactic proof-based approach introduced here. In order to do that, we will need to put the proof system to use and actually prove some theorems in hybrid logic. To make our theorem-proving life easier, we will first derive some additional properties of the proof system and we will extend it with some handy new inference rules. The first of these properties is the deduction theorem:

Theorem 2.3.2 (Deduction)**.** Let φ, ψ be $\mathscr{L}(\forall)$ -formulas and Γ a set of $\mathscr{L}(\forall)$ -formulas. Then:

$$\Gamma \vdash \varphi \rightarrow \psi \iff \Gamma \cup \{\varphi\} \vdash \psi$$

Proof. " $\Longrightarrow$ " By definition, if $\Gamma \vdash \varphi \rightarrow \psi$, then there exist some formula $\chi = \chi_1 \wedge \chi_2 \wedge ... \wedge \chi_n$, with $n \in \mathbb{N}$ and each $\chi_i \in \Gamma$, such that $\vdash \chi \rightarrow (\varphi \rightarrow \psi)$. We apply the Tautology 13 and unfold χ, obtaining: $\vdash (\chi_1 \wedge ... \wedge \chi_n \wedge \varphi) \rightarrow \psi$. Now, $\chi_1, ..., \chi_n$ and φ are all formulas in $\Gamma \cup \{\varphi\}$, so by definition we find that $\Gamma \cup \{\varphi\} \vdash \psi$.

" $\Longleftarrow$ " Suppose now $\Gamma \cup \{\varphi\} \vdash \psi$. Let $\chi = \chi_1 \wedge \chi_2 \wedge ... \chi_n$, for $\chi_i \in \Gamma \cup \{\varphi\}$ and $n \in \mathbb{N}$, such that $\vdash \chi \rightarrow \psi$. We can assume without loss of generality that all χ_i's are distinct. By the rules of propositional calculus, if $\vdash \chi \rightarrow \psi$, then $\vdash (\chi \wedge \varphi) \rightarrow \psi$. If φ is not among the χ_i's, we apply Tautology 13, obtaining $\vdash \chi \rightarrow (\varphi \rightarrow \psi)$. This completes our proofs, since χ is then a conjunction of formulas in Γ. Otherwise, if there is some i such that $\chi_i = \varphi$, we have

$\vdash (\chi_1 \wedge ... \chi_{i-1} \wedge \varphi \wedge \chi_{i+1} \wedge ... \wedge \chi_n) \rightarrow \psi$. Since conjunction is associative and commutative, we can move φ to the right and then apply Tautology 13, obtaining: $\vdash \theta \rightarrow (\varphi \rightarrow \psi)$, where $\theta := \chi_1 \wedge ... \wedge \chi_{i-1} \wedge \chi_{i+1} \wedge ... \wedge \chi_n$. Now, φ does not occur in θ, otherwise the formulas in χ would no longer be distinct. Thus all formulas in θ belong to Γ. So we have shown that $\Gamma \vdash \varphi \rightarrow \psi$. □

Proposition 2.3.3. The following are true:

(i) $\vdash \varphi$ iff $\emptyset \vdash \varphi$

(ii) If $\Gamma \vdash \varphi$ and $\Gamma \subseteq \Delta$, then $\Delta \vdash \varphi$. *(Monotonicity)*

(iii) $\vdash \varphi$ iff for all Γ, $\Gamma \vdash \varphi$

Proof of (i). Immediate, by the definition of logical consequence. The only subset of $\emptyset$ is $\emptyset$, and so $\emptyset \vdash \varphi$ is definitionally the same as $\vdash \varphi$. □

Proof of (ii). Since $\Gamma \subseteq \Delta$, any subset of Γ is a subset of Δ, so any finite conjunction χ of formulas in Γ such that $\vdash \chi \rightarrow \varphi$ is also a finite conjunction of formulas in Δ. □

Proof of (iii). Immediate, by (i), (ii); since $\emptyset \subseteq \Gamma$ for all Γ. □

The following theorem will also help smoothen our proofs:

Theorem 2.3.4 (Renaming Bound Variables)**.** Let φ be some $\mathscr{L}(\forall)$ -formula. If y does not occur in φ and y is substitutable for x in φ, then:

$$\vdash (\forall x\ \varphi) \leftrightarrow (\forall y\ \varphi\ [y/x])$$

Proof. We first prove that $\vdash (\forall x\ \varphi) \rightarrow (\forall y\ \varphi\ [y/x])$:

(1) $\vdash \forall x\ \varphi \rightarrow \varphi\ [y/x]$ *(Axiom Q2; y is subst. for x in φ)*

(2) $\vdash \forall y\ (\forall x\ \varphi \rightarrow \varphi\ [y/x])$ *(Generalize on (1))*

(3) $\vdash (\forall y\ (\forall x\ \varphi \rightarrow \varphi\ [y/x])) \rightarrow (\forall x\ \varphi \rightarrow \forall y\ \varphi\ [y/x])$ *(Axiom Q1)*

(4) $\vdash \forall x\ \rightarrow \forall y\ \varphi\ [y/x]$ *(MP (3), (2))*

Next, the other direction: $\vdash (\forall y\ \varphi\ [y/x]) \to (\forall x\ \varphi)$. Notice that both conditions required for the Re-Replacement Lemma (Lemma 2.1.3) hold by hypothesis.

(1) $\vdash \forall y\ \varphi\ [y/x] \to \varphi\ [y/x][x/y]$ *(Axiom Q2, using Lemma 2.1.3)*

(2) $\vdash \forall y\ \varphi\ [y/x] \to \varphi$ *(Rewrite (1) by Lemma 2.1.3)*

(3) $\vdash \forall x\ (\forall y\ \varphi\ [y/x] \to \varphi)$ *(Generalize on (2))*

(4) $\vdash \forall x\ (\forall y\ \varphi\ [y/x] \to \varphi) \to (\forall y\ \varphi\ [y/x] \to \forall x\ \varphi)$ *(Axiom Q1)*

(5) $\vdash \forall y\ \varphi\ [y/x] \to \forall x\ \varphi$ *(MP (4), (3))*

At step 4, we used Axiom Q1, which rests on the fact that x does not occur free in $\varphi\ [y/x]$. This is indeed true, but should not go without a small comment. We saw in Proposition 2.1.1 that there are no free occurences of x in $\varphi\ [y/x]$ if $x \neq y$, but what about the case that $x = y$? Thankfully we know by hypothesis that y does not occur in φ; so if $x = y$, we know that x does not occur in φ. So clearly there is no (free) occurrence of x in $\varphi\ [y/x]$ either, and we can safely use Axiom Q1.

□

Corollary 2.3.5. Let φ be some $\mathscr{L}(\forall)$ -formula. If y does not occur in φ and y is substitutable for x in φ, then:

$$\vdash (\exists x\ \varphi) \leftrightarrow (\exists y\ \varphi\ [y/x])$$

Proof. By the definition of the existential binder, this follows immediately from the main theorem, using Tautology 12. □

2.3.1 Some Formal Derivations

Now we are able to define new inference rules and prove their correctness. Their addition provides no additional proof power to our system. Their only point is to ensure proofs are kept at a manageable length, by packing together commonly needed sequences of proof steps under a single name. Let φ, ψ be $\mathscr{L}(\forall)$ -formulas and Γ a set of $\mathscr{L}(\forall)$ -formulas. The following statements are true:

(i) (Premise) If $\varphi \in \Gamma$, then $\Gamma \vdash \varphi$

(ii) (MP) If $\Gamma \vdash (\varphi \rightarrow \psi)$ and $\Gamma \vdash \varphi$, then $\Gamma \vdash \psi$

(iii) (Contraposition) If $\Gamma \vdash (\varphi \rightarrow \psi)$, then $\Gamma \vdash \neg\psi \rightarrow \neg\varphi$

(iv) (Contraposition) If $\Gamma \vdash (\neg\varphi \rightarrow \neg\psi)$, then $\Gamma \vdash \psi \rightarrow \varphi$

(v) (DNI) $\Gamma \vdash \varphi$ implies $\Gamma \vdash \neg\neg\varphi$

(vi) (DNE) $\Gamma \vdash \neg\neg\varphi$ implies $\Gamma \vdash \varphi$

(vii) (Conj. Intro.) $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$ implies $\Gamma \vdash (\varphi \wedge \psi)$

(viii) (Conj. Elim.) $\Gamma \vdash (\varphi \wedge \psi)$ implies $\Gamma \vdash \varphi$ and $\Gamma \vdash \psi$

(ix) (Universal Intro.) If x does not occur free in any formula in Γ, then

$$\Gamma \vdash \varphi \text{ implies } \Gamma \vdash \forall x\ \varphi$$

(x) (Universal Elim.) $\Gamma \vdash \forall x\ \varphi$ implies $\Gamma \vdash \varphi$

Proof of (i). Since $\varphi \rightarrow \varphi$ is a tautology, we have $\vdash \varphi \rightarrow \varphi$, so by Proposition 2.3.3.(iii), $\Gamma \vdash \varphi \rightarrow \varphi$. Applying deduction, we get $\Gamma \cup \{\varphi\} \vdash \varphi$. And since $\varphi \in \Gamma$, we know $\Gamma \cup \{\varphi\} = \Gamma$. So $\Gamma \vdash \varphi$. $\square$

Proof of (ii). By definition, we know there exists χ_1 and χ_2 conjunctions of formulas from Γ, such that $\vdash \chi_1 \rightarrow (\varphi \rightarrow \psi)$ and $\vdash \chi_2 \rightarrow \varphi$. Now let $\chi_3 = \chi_1 \wedge \chi_2$. Applying Tautology 7, it follows that $\vdash \chi_3 \rightarrow (\varphi \rightarrow \psi)$ and $\vdash \chi_3 \rightarrow \varphi$. Now we apply Tautology 8 twice and reach $\vdash \chi_3 \rightarrow \psi$, and thus $\Gamma \vdash \psi$. $\square$

Proof of (iii) *and* (iv). By Tautology 2, using Proposition 2.3.3.(iii) and MP (rule (ii)). $\square$

Proof of (v) *and* (vi). By Tautology 4, using Proposition 2.3.3.(iii) and MP. $\square$

Proof of (vii) *and* (viii). By Tautologies 5 and 6, using Proposition 2.3.3.(iii) and MP. $\square$

Proof of (ix). Let χ be a conjunction of formulas in Γ such that $\vdash \chi \rightarrow \varphi$. Then:

(1) $\vdash \chi \to \varphi$ *(Assumption)*

(2) $\vdash \forall x\,(\chi \to \varphi)$ *(Generalize (1))*

(3) $\vdash \forall x\,(\chi \to \varphi) \to (\chi \to \forall x\ \varphi)$ *(Ax. Q1; x is not free in χ)*

(4) $\vdash \chi \to \forall x\ \varphi$ *(MP (3), (2))*

(5) $\Gamma \vdash \forall x\ \varphi$ *(Definition of syntactic consequence)*

□

Proof of (x). Let z be a fresh variable relative to $\forall x\ \varphi$; i.e., one that is not displayed at all. We know z is substitutable for x in φ and z is not free in φ. Then:

(1) $\Gamma \vdash \forall x\ \varphi$ *(Assumption)*

(2) $\Gamma \vdash \forall x\ \varphi \to \forall z\ \varphi\,[z/x]$ *(Thm. 2.3.4 and Prop. 2.3.3.(iii))*

(3) $\Gamma \vdash \forall z\ \varphi\,[z/x]$ *(MP (2), (1))*

(4) $\Gamma \vdash \forall z\ \varphi\,[z/x] \to \varphi\,[z/x][x/z]$ *(Ax. Q2 [safe, by L. 2.1.3] and P. 2.3.3.(iii))*

(5) $\Gamma \vdash \varphi\,[z/x][x/z]$ *(MP (3), (2))*

(6) $\Gamma \vdash \varphi$ *(Rewrite (4) by L. 2.1.3)*

□

Proposition 2.3.6. $\vdash \varphi \to \psi$ implies $\vdash \Diamond\,\varphi \to \Diamond\,\psi$

Proof. We have:

(1) $\vdash \varphi \to \psi$ *(Assumption)*

(2) $\vdash (\varphi \to \psi) \to (\neg\psi \to \neg\varphi)$ *(Tautology 2)*

(3) $\vdash \neg\psi \to \neg\varphi$ *(MP (2), (1))*

(4) $\vdash \Box(\neg\psi \to \neg\varphi)$ *(Generalize (3))*

(5) $\vdash \Box(\neg\psi \to \neg\varphi) \to (\Box\neg\psi \to \Box\neg\varphi)$ *(Ax. K)*

(6) $\vdash \Box\neg\psi \to \Box\neg\varphi$ *(MP (5), (4))*

(7) $\vdash (\Box\neg\psi \to \Box\neg\varphi) \to (\neg\Box\neg\varphi \to \neg\Box\neg\psi)$ *(Tautology 2)*

(8) $\vdash \neg\Box\neg\varphi \to \neg\Box\neg\psi$ *(MP (7), (6))*

(9) $\vdash \Diamond\varphi \to \Diamond\psi$ *(Rewrite (8) by definition of diamond)*

□

Proposition 2.3.7. For all formulas φ, ψ, the following are theorems:

(i) $\vdash (\varphi \to \exists x\, \psi) \to \exists x\, (\varphi \to \psi)$

(ii) $\vdash (\varphi \wedge \exists x\, \psi) \to \exists x\, (\varphi \wedge \psi)$, if x does not occur free in φ

(iii) $\vdash \forall x\, (\varphi \to \psi) \to (\forall x\, \varphi \to \forall x\, \psi)$

(iv) $\vdash \Diamond\varphi \to \exists y\, \Diamond((\exists x\, \psi \to \psi\, [y/x]) \wedge \varphi)$, if y is not free in either φ or ψ, and y is substitutable for x in ψ

(v) $\vdash (\Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n) \to \Box(\varphi_1 \wedge \ldots \wedge \varphi_n)$, for all $n \geq 1$

Proof of (i). Let $\Gamma = \{\neg\exists x\, (\varphi \to \psi)\}$. We have:

(1) $\Gamma \vdash \neg\exists x\, (\varphi \to \psi)$ *(Premise)*

(2) $\Gamma \vdash \neg\neg\forall x\, \neg(\varphi \to \psi)$ *(Rewrite (1) by definition of existential)*

(3) $\Gamma \vdash \forall x\, \neg(\varphi \to \psi)$ *(DNE (2))*

(4) $\Gamma \vdash \neg(\varphi \to \psi)$ *(Univ. Elim. (3))*

(5) $\Gamma \vdash \neg(\varphi \to \psi) \to (\varphi \wedge \neg\psi)$ *(Tautology 10 and Prop. 2.3.3.(iii))*

(6) $\Gamma \vdash \varphi \wedge \neg\psi$ *(MP (5), (4))*

(7) $\Gamma \vdash \varphi$ *(Conj. Elim. (6))*

(8) $\Gamma \vdash \neg\psi$ *(Conj. Elim. (6))*

(9) $\Gamma \vdash \forall x\, \neg\psi$ *(Univ. Intro. (8))*

(10) $\Gamma \vdash \neg\neg\forall x\, \neg\psi$ *(DNI (9))*

(11) $\Gamma \vdash \neg\exists x\, \psi$ *(Rewrite (10) by definition of existential)*

(12) $\Gamma \vdash \varphi \wedge \neg\exists x\, \psi$ *(Conj. Intro. (7) and (11))*

(13) $\Gamma \vdash (\varphi \wedge \neg\exists x\, \psi) \to \neg(\varphi \to \exists x\, \psi)$ *(Tautology 10 and Prop. 2.3.3.(iii))*

(14) $\Gamma \vdash \neg(\varphi \to \exists x\, \psi)$ *(MP (13), (12))*

(15) $\vdash \neg\exists x\, (\varphi \to \psi) \to \neg(\varphi \to \exists x\, \psi)$ *(Deduction (14))*

(16) $\vdash (\varphi \to \exists x\, \psi) \to \exists x\, (\varphi \to \psi)$ *(Contraposition (15))*

□

Proof of (ii). Let $\Gamma = \{\neg\exists x\, (\varphi \wedge \psi)\}$. We have:

(1) $\Gamma \vdash \neg\neg\forall x\, \neg(\varphi \wedge \psi)$ *(Premise and rewrite by existential)*

(2) $\Gamma \vdash \forall x\, \neg(\varphi \wedge \psi)$ *(DNE (1))*

(3) $\Gamma \vdash \neg(\varphi \wedge \psi)$ *(Univ. Elim. (2))*

(4) $\Gamma \vdash \neg(\varphi \wedge \psi) \to (\varphi \to \neg\psi)$ *(Tautology 11 and Prop. 2.3.3.(iii))*

(5) $\Gamma \vdash \varphi \to \neg\psi$ *(MP (3), (2))*

(6) $\Gamma \vdash \forall x\, (\varphi \to \neg\psi)$ *(Univ. Intro. (5))*

(7) $\Gamma \vdash \forall x\, (\varphi \to \neg\psi) \to (\varphi \to \forall x\, \neg\psi)$ *(Ax. Q1 and Prop. 2.3.3.(iii))*

(8) $\Gamma \vdash \varphi \to \forall x\, \neg\psi$ *(MP (7), (6))*

(9) $\Gamma \cup \{\varphi\} \vdash \forall x\, \neg\psi$ *(Deduction (8))*

(10) $\Gamma \cup \{\varphi\} \vdash \neg\neg\forall x\, \neg\psi$ *(DNI (9))*

(11) $\Gamma \vdash \varphi \to \neg\exists x\, \psi$ *(Deduction (10) and rewrite by existential)*

(12) $\Gamma \vdash (\varphi \to \neg\exists x\, \psi) \to \neg(\varphi \wedge \exists x\, \psi)$ *(Tautology 11 and Prop. 2.3.3.(iii))*

(13) $\Gamma \vdash \neg(\varphi \wedge \exists x\, \psi)$ *(MP (12), (11))*

(14) $\vdash \neg\exists x\, (\varphi \wedge \psi) \to \neg(\varphi \wedge \exists x\, \psi)$ *(Deduction (13))*

(15) $\vdash (\varphi \wedge \exists x\, \psi) \to \exists x\, (\varphi \wedge \psi)$ *(Contraposition (14))*

$\square$

Proof of (iii). Let $\Gamma = \{\forall x\,(\varphi \to \psi), \forall x\ \varphi\}$. We have:

(1) $\Gamma \vdash \forall x\,(\varphi \to \psi)$ *(Premise)*

(2) $\Gamma \vdash \forall x\ \varphi$ *(Premise)*

(3) $\Gamma \vdash \varphi \to \psi$ *(Univ. Elim. (1))*

(4) $\Gamma \vdash \varphi$ *(Univ. Elim. (2))*

(5) $\Gamma \vdash \psi$ *(MP (3),(2))*

(6) $\Gamma \vdash \forall x\ \psi$ *(Univ. Intro. (5))*

(7) $\{\forall x\,(\varphi \to \psi)\} \vdash \forall x\ \varphi \to \forall x\ \psi$ *(Deduction (6))*

(8) $\vdash \forall x\,(\varphi \to \psi) \to (\forall x\ \varphi \to \forall x\ \psi)$ *(Deduction (7))*

$\square$

Proof of (iv). We have:

(1) $\vdash \exists x\ \psi \to \exists y\ \psi\,[y/x]$ *(Corollary 2.3.5)*

(2) $\vdash (\exists x\ \psi \to \exists y\ \psi\,[y/x]) \to \exists y\,(\exists x\ \psi \to \psi\,[y/x])$ *(Prop. 2.3.7.(i))*

(3) $\vdash \exists y\,(\exists x\ \psi \to \psi\,[y/x])$ *(MP (2), (1))*

(4) $\{\varphi\} \vdash \exists y\,(\exists x\ \psi \to \psi\,[y/x])$ *(Prop. 2.3.3.(iii) and (3))*

(5) $\{\varphi\} \vdash \varphi$ *(Premise)*

(6) $\{\varphi\} \vdash \exists y\,(\exists x\ \psi \to \psi\,[y/x]) \wedge \varphi$ *(Conj. Intro. (4), (5))*

(7) $\{\varphi\} \vdash (\exists y\ (\exists x\ \psi \to \psi\ [y/x]) \wedge \varphi) \to \exists y\ (\varphi \wedge (\exists x\ \psi \to \psi\ [y/x]))$ *(Prop. 2.3.7.(ii) and Prop. 2.3.3.(iii))*

(8) $\{\varphi\} \vdash \exists y\,((\exists x\ \psi \to \psi\,[y/x]) \wedge \varphi)$ *(MP (7), (6))*

(9) $\vdash \varphi \to \exists y\,((\exists x\ \psi \to \psi\,[y/x]) \wedge \varphi)$ *(Deduction (8))*

(10) $\vdash \Diamond\varphi \to \Diamond\exists y\,((\exists x\ \psi \to \psi\,[y/x]) \wedge \varphi)$ *(Prop. 2.3.6 and (9))*

(11) $\{\Diamond\varphi\} \vdash \Diamond\exists y\,((\exists x\,\psi \to \psi\,[y/x]) \wedge \varphi)$ *(Deduction (10))*

(12) $\{\Diamond\varphi\} \vdash \Diamond\exists y\,((\exists x\,\psi \to \psi\,[y/x]) \wedge \varphi) \to \exists y\,\Diamond((\exists x\,\psi \to \psi\,[y/x]) \wedge \varphi)$ *((Contrapositive of Ax. Barcan))*

(13) $\{\Diamond\varphi\} \vdash \exists y\,\Diamond((\exists x\,\psi \to \psi\,[y/x]) \wedge \varphi)$ *(MP (12), (13))*

(14) $\vdash \Diamond\varphi \to \exists y\,\Diamond((\exists x\,\psi \to \psi\,[y/x]) \wedge \varphi)$ *(Deduction (13))*

□

Proof of (v). We will do induction on n. The base case is $\vdash \varphi_1 \to \varphi_1$, which is an instance of Tautology 1. We will treat $n = 2$ separately, as we will need it for the inductive step: $\vdash (\Box\varphi_1 \wedge \Box\varphi_2) \to \Box(\varphi_1 \wedge \varphi_2)$. Its derivation is as follows:

(1) $\vdash \varphi_1 \to (\varphi_2 \to (\varphi_1 \wedge \varphi_2))$ *(Tautology 5)*

(2) $\vdash \Box(\varphi_1 \to (\varphi_2 \to (\varphi_1 \wedge \varphi_2)))$ *(Generalize (1))*

(3) $\vdash \Box(\varphi_1 \to (\varphi_2 \to (\varphi_1 \wedge \varphi_2))) \to (\Box\varphi_1 \to \Box(\varphi_2 \to (\varphi_1 \wedge \varphi_2)))$ *(Axiom K)*

(4) $\vdash \Box\varphi_1 \to \Box(\varphi_2 \to (\varphi_1 \wedge \varphi_2))$ *(MP (3), (2))*

(5) $\{\Box\varphi_1\} \vdash \Box(\varphi_2 \to (\varphi_1 \wedge \varphi_2))$ *(Deduction (4))*

(6) $\{\Box\varphi_1\} \vdash \Box(\varphi_2 \to (\varphi_1 \wedge \varphi_2)) \to (\Box\varphi_2 \to \Box(\varphi_1 \wedge \varphi_2))$ *(Axiom K)*

(7) $\{\Box\varphi_1\} \vdash \Box\varphi_2 \to \Box(\varphi_1 \wedge \varphi_2)$ *(MP (5), (4))*

(8) $\vdash \Box\varphi_1 \to (\Box\varphi_2 \to \Box(\varphi_1 \wedge \varphi_2))$ *(Deduction (6))*

(9) $\vdash (\Box\varphi_1 \to (\Box\varphi_2 \to \Box(\varphi_1 \wedge \varphi_2))) \to (\Box\varphi_1 \wedge \Box\varphi_2) \to \Box(\varphi_1 \wedge \varphi_2)$ *(T. 13)*

(10) $\vdash (\Box\varphi_1 \wedge \Box\varphi_2) \to \Box(\varphi_1 \wedge \varphi_2)$ *(MP (9), (8))*

For the inductive step, let $\Gamma = \{\Box\varphi_{n+1}, \Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n\}$. We have:

(1) $\Gamma \vdash (\Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n) \to \Box(\varphi_1 \wedge \ldots \wedge \varphi_n)$ *(Induction hypothesis)*

(2) $\Gamma \vdash \Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n$ *(Premise)*

(3) $\Gamma \vdash \Box(\varphi_1 \wedge \ldots \wedge \varphi_n)$ *(MP (1), (2))*

(4) $\Gamma \vdash \Box\varphi_{n+1}$ *(Premise)*

(5) $\Gamma \vdash \Box(\varphi_1 \wedge \ldots \wedge \varphi_n) \wedge \Box\varphi_{n+1}$ *(Conj. Intro. (3), (4))*

(6) $\Gamma \vdash (\Box(\varphi_1 \wedge \ldots \wedge \varphi_n) \wedge \Box\varphi_{n+1}) \to \Box(\varphi_1 \wedge \ldots \wedge \varphi_n \wedge \varphi_{n+1})$ *(Case $n = 2$)*

(7) $\Gamma \vdash \Box(\varphi_1 \wedge \ldots \wedge \varphi_n \wedge \varphi_{n+1})$ *(MP (6), (5))*

(8) $\{\Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n\} \vdash \Box\varphi_{n+1} \to \Box(\varphi_1 \wedge \ldots \wedge \varphi_n \wedge \varphi_{n+1})$ *(Deduction (7))*

(9) $\vdash (\Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n) \to (\Box\varphi_{n+1} \to \Box(\varphi_1 \wedge \ldots \wedge \varphi_n \wedge \varphi_{n+1}))$ *(Ded. (8))*

(10) $\vdash (\Box\varphi_1 \wedge \ldots \wedge \Box\varphi_n \wedge \Box\varphi_{n+1}) \to \Box(\varphi_1 \wedge \ldots \wedge \varphi_n \wedge \varphi_{n+1})$ *(T. 13)*

□

We end this section by proving two additional results that we will need in our proof of *completeness*.

Proposition 2.3.8. Let Γ be a set of formulas and φ some formula. $\Gamma \nvdash \varphi$ if and only if $\Gamma \cup \{\neg\varphi\}$ is consistent.

Proof. " $\Longrightarrow$ " Assume $\Gamma \nvdash \varphi$. Suppose, furthermore, that

$$\begin{aligned}
\Gamma \cup \{\neg\varphi\} \text{ is inconsistent} &\iff \Gamma \cup \{\neg\varphi\} \vdash \bot && \textit{(Definition)}\\
&\iff \Gamma \vdash \neg\varphi \to \bot && \textit{(Deduction Theorem)}\\
&\iff \Gamma \vdash \neg\neg\varphi && \textit{(Definition)}\\
&\iff \Gamma \vdash \varphi && \textit{(DNE)}
\end{aligned}$$

But, by assumption, $\Gamma \nvdash \varphi$. We have reached a contradiction. We conclude, then, that $\Gamma \cup \{\neg\varphi\}$ is consistent.

" $\Longleftarrow$ " By the same reasoning, we have that

$$\begin{aligned}
\Gamma \cup \{\neg\varphi\} \text{ is consistent} &\iff \Gamma \cup \{\neg\varphi\} \nvdash \bot && \textit{(Definition)}\\
&\iff \Gamma \nvdash \neg\varphi \to \bot && \textit{(Deduction Theorem)}\\
&\iff \Gamma \nvdash \neg\neg\varphi && \textit{(Definition)}\\
&\iff \Gamma \nvdash \varphi && \textit{(DNE)}
\end{aligned}$$

□

Lastly, using one of the theorems we proved under Proposition 2.3.7, we are able to give a corollary to Theorem 2.3.4. Recall, from the first chapter, the definition of free variants (Definition 2.1.6). We prove the following:

Corollary 2.3.9. Let φ be some $\mathscr{L}(\forall)$ -formula and φ' its x-free variant. Then $\vdash \varphi \leftrightarrow \varphi'$.

Proof. The proof is by induction on formulas. The only case that requires slightly non-trivial work is for $\forall x\ \varphi$. Suppose, by induction, that $\vdash \varphi \leftrightarrow \varphi'$. We have to show that $\vdash (\forall x\ \varphi) \leftrightarrow (\forall v\ \varphi'\ [v/x])$, where v is a variable that does not occur at all in φ' and differs from x. Thanks to Proposition 2.3.7.(iii) and some propositional calculus, by generalizing on the induction hypothesis we arrive at $\vdash (\forall x\ \varphi) \leftrightarrow (\forall x\ \varphi')$. And by the main theorem, we know that $\vdash (\forall x\ \varphi') \leftrightarrow (\forall v\ \varphi'\ [v/x])$. Replacing $\forall x\ \varphi'$ by its equivalent, we reach $\vdash (\forall x\ \varphi) \leftrightarrow (\forall v\ \varphi'\ [v/x])$, as desired.

All other cases follow from the definition of free variants. □

Chapter 3

Soundness

In this section, we prove the first result that links the semantics of hybrid logic to its proof system: soundness. We will show that all theorems of hybrid logic are valid in all standard models. In doing so, we will follow the method outlined in [5]. Since soundness proofs are seldom spelled out in complete detail, we will strive here to provide an explanation of all the steps involved. Our treatment of tautologies is adapted from [15].

Definition 3.0.1 (Soundness)**.** The hybrid logic of $\mathscr{H}(\forall)$ is **sound** with respect to the class of standard models if and only if for all sets of formulas Γ and all formulas φ:

$$\Gamma \vdash \varphi \text{ implies } \Gamma \vDash \varphi$$

To prove this implication, we will show all axioms are valid, and all rules of inference preserve validity. Let us begin with the tautologies. We notice there is a very straightforward way to define a propositional evaluation based on the satisfaction relation:

Definition 3.0.2. Let $\mathscr{M}$ be a model, s a state and g and assignment function. The evaluation associated to $\mathscr{M}, \mathrm{s}, \mathrm{g}$ is the function $e_{Sat} : Form_{\mathscr{L}(\forall)} \to \{0,1\}$,

$$e_{Sat}(\varphi) = 1 \iff \mathscr{M}, \mathrm{s}, \mathrm{g} \vDash \varphi$$

It should come as no surprise that e_{Sat} fulfills the properties of an evaluation function:

Proposition 3.0.1. The function e_{Sat} is a propositional evaluation.

Proof.

1. $e_{Sat}(\bot) = 1$ iff $\mathscr{M}, \mathrm{s}, \mathrm{g} \vDash \bot$ iff **False**, so $e_{Sat}(\bot) = 0$
2. $e_{sat}(\varphi \to \psi) = 1$ iff $(\mathscr{M}, \mathrm{s}, \mathrm{g} \vDash \varphi$ implies $\mathscr{M}, \mathrm{s}, \mathrm{g} \vDash \psi)$ iff $(e_{Sat}(\varphi) = 1$ implies $e_{Sat}(\psi) = 1)$

□

Clearly, then:

Lemma 3.0.2. Let φ be any formula. If φ is a tautology, then φ is valid.

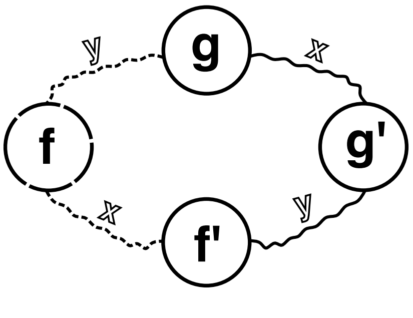
Proof. Let $\mathscr{M}$ be a model, s a state in $\mathscr{M}$, and g an assignment, and let e_{Sat} be the evaluation corresponding to $\mathscr{M}, \mathrm{s}, \mathrm{g}$. Since φ is a tautology, it is true under all evaluations, so $e_{Sat}(\varphi) = 1$. By the definition of e_{Sat}, this means $\mathscr{M}, \mathrm{s}, \mathrm{g} \vDash \varphi$. □

Lemma 3.0.3. Let g, g' and f' be assignment functions such that g' is an x-variant of g, and f' is a y-variant of g'. Then there exists an assignment function f, such that f is a y-variant of g and f is an x-variant of g'.

Proof. If $x = y$, then g itself has the required properties.

If $x \neq y$, let $f(v) = \begin{cases} f'(v), & \text{if } v \neq x \\ g(v), & \text{otherwise} \end{cases}$. By definition, f is an x-variant of f'. To show that f is a y-variant of g, note that if $v \neq x$ and $v \neq y$, $f'(v) = g(v)$. Rewriting the definition of f by this identity, we get: $f(v) = \begin{cases} g(v), & \text{if } v \neq y \\ f'(v), & \text{otherwise} \end{cases}$. It follows that f is a y-variant of g. □

The dotted lines in the figure below indicate the function we defined and the relations we showed it bears to the other functions.



Lemma 3.0.4. Let φ be a formula, and x a state variable that does not occur free in φ. Then $\models \varphi \leftrightarrow \forall x\ \varphi$.

Proof. Take any model, state and assignment function $\mathscr{M}, \mathrm{s}, \mathrm{g}$. We have to prove that $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \varphi$ iff $\mathscr{M}, \mathrm{s}, \mathrm{g}' \models \varphi$, for all x-variants g' of g.

For the right-to-left direction, notice g is itself an x-variant of g, so it follows immediately that $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \varphi$.

In the other direction, we will do induction on φ. The implication is trivially true if φ is $\bot$, a propositional variable or a nominal, because their satisfaction does not depend on assignment functions. If φ is a state variable y, notice that since we know x is not free in φ, it follows that $y \neq x$. So $g'(y) = g(y)$, and thus $\mathscr{M}, \mathrm{s}, \mathrm{g}' \models \varphi$. The boolean and modal cases, $\varphi \rightarrow \psi$ and $\Box\varphi$, follow directly from the induction hypotheses.

Finally, consider $\forall y\ \varphi$. Since x is not free in $\forall y\ \varphi$, either $x = y$, or x is not free in φ. If $x = y$, we have to show that $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \forall x\ \varphi$ implies $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \forall x \forall x\ \varphi$. This is evidently the case: being a variant of an assignment function is a transitive relation, so it makes no difference how many times we write the binder at the start of a formula. Next, suppose x is not free in φ and suppose $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \forall y\ \varphi$. We have to show that $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \forall x \forall y\ \varphi$. Since, as in first order logic, binding is commutative (because an x-variant of a y-variant of g is also a y-variant of an x-variant of g), our goal is equivalent to $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \forall y \forall x\ \varphi$. So let g' be a y-variant of g. By assumption, then, $\mathscr{M}, \mathrm{s}, \mathrm{g}' \models \varphi$. We apply the induction hypothesis to this relation and obtain $\mathscr{M}, \mathrm{s}, \mathrm{g}' \models \forall x\ \varphi$, as desired. □

Lemma 3.0.5. Let $\mathscr{M} = \langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$ be a model, $\mathrm{s} \in \mathrm{W}$, and g an assignment. Let, furthermore, x and y be two variables such that y is substitutable for x in φ, and g' is the x-variant of g with $\mathrm{g}'(x) = \mathrm{g}(y)$. Then:

$$\mathscr{M}, s, g \models \varphi\ [y/x] \text{ iff } \mathscr{M}, s, g' \models \varphi$$

Proof. For the $\bot$, nominal and propositional cases, $\varphi\ [y/x] = \varphi$ and satisfaction does not depend on the assignment function, so the equivalence holds trivially.

Suppose now $\varphi = z$, for some variable $z \neq x$. Then $\varphi\ [y/x] = \varphi = z$, and since g and g' agree to the assignment given to z, the equivalence holds. If $\varphi = x$, we have to show that $\mathscr{M}, \mathrm{s}, \mathrm{g} \models y$ iff $\mathscr{M}, \mathrm{s}, \mathrm{g}' \models x$. Since g'($x$) = g($y$), this is clearly also the case.

The boolean and modal cases, $\varphi \to \psi$ and $\Box \varphi$, follow directly from the induction hypotheses.

What requires a bit more work is the bound case, $\forall v\ \varphi$. The interesting case is when x is free in φ, and all of v, x and y are different variables. But let us begin by treating the other cases briefly. If x does not occur free in φ, we know by Proposition 2.1.2 that $\varphi\ [y/x] = \varphi$, so we have to show $\mathscr{M}, \mathrm{s}, \mathrm{g} \vDash \varphi$ iff $\mathscr{M}, \mathrm{s}, \mathrm{g}' \vDash \varphi$. Both directions of this equivalence follow from Lemma 3.0.4. So suppose x does occur free in φ. If $v = x$, we have to show $\mathscr{M}, \mathrm{s}, \mathrm{g} \vDash \forall x\ \varphi$ iff $\mathscr{M}, \mathrm{s}, \mathrm{g}' \vDash \forall x\ \varphi$. Again, both directions of this equivalence follow from Lemma 3.0.4. If, otherwise, $v \neq x$, then clearly also $v \neq y$. Since if $v = y$, we would know by hypothesis that y is substitutable for x in $\forall y\ \varphi$, and also that x is free in φ. By the definition of substitutability, this is contradictory.

Consider now $v \neq x$ and $v \neq y$, and x occurs free in φ. Here is where Lemma 3.0.3 becomes useful. We prove the left-to-right direction first. After applying substitution and unfolding the definition of satisfaction, our goal becomes:

$$\mathscr{M}, \mathrm{s}, \mathrm{g} \vDash \forall v\ (\varphi\ [y/x]) \text{ implies } \mathscr{M}, \mathrm{s}, f' \vDash \varphi, \text{ for all v-variants } f' \text{ of } g'$$

So let f' be a v-variant of g'. Since g' is itself an x-variant of g, Lemma 3.0.3 tells us that there exists a function f which is a v-variant of g and an x-variant of f'. Thus, from $\mathscr{M}, \mathrm{s}, \mathrm{g} \vDash \forall v\ (\varphi\ [y/x])$, we conclude that $\mathscr{M}, \mathrm{s}, f \vDash \varphi\ [y/x]$. We also notice that $f'(x) = f(y)$; because $\mathrm{g}'(x) = \mathrm{g}(y)$ and, by being variants on other variables, $f'(x) = \mathrm{g}'(x)$ and $f(y) = \mathrm{g}(y)$. So we meet all the requirements in order to apply the induction hypothesis to f and f', and obtain $\mathscr{M}, \mathrm{s}, f' \vDash \varphi$.

The right-to-left direction follows the same reasoning. Our goal is:

$$\mathscr{M}, \mathrm{s}, \mathrm{g}' \vDash \forall v\ \varphi \text{ implies } \mathscr{M}, \mathrm{s}, f \vDash \varphi\ [y/x], \text{ for all v-variants } f \text{ of } g$$

So let f be a v-variant of g, and let f' be the x-variant of f which is also a v-variant of g'. We know f' exists via Lemma 3.0.3 Since $\mathscr{M}, \mathrm{s}, f' \vDash \varphi$ and $f'(x) = f(y)$, we apply the induction hypothesis to get $\mathscr{M}, \mathrm{s}, f \vDash \varphi\ [y/x]$. $\Box$

Lemma 3.0.6. Let $\mathscr{M} = \langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$ be a model, $\mathrm{s} \in \mathrm{W}$, and g an assignment. Let, furthermore, x be a variable and i a nominal such that g' is the x-variant of g with $g'(x) = V(i)$. Then:

$$\mathscr{M}, s, g \vDash \varphi\ [i/x] \text{ iff } \mathscr{M}, s, g' \vDash \varphi$$

Proof. By induction on φ, once again. The steps required for this proof are virtually identical to the ones we followed in the previous proof, so we will not bore the reader by repeating the details. □

Theorem 3.0.7 (Weak Soundness)**.** Let φ be any formula. Then:

$$\vdash \varphi \text{ implies } \vDash \varphi$$

Proof. We will do induction on $\vdash \varphi$.

(1) *(T)* We have already proved this case, under Lemma 3.0.2.

(2) *(K)* We want to show that $\vDash \Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$. So suppose $\mathscr{M}, s, g \vDash \Box(\varphi \to \psi)$ and $\mathscr{M}, s, g \vDash \Box\varphi$, for an arbitrary choice of $\mathscr{M}, s, g$. Then, in all states s' accessible from s via R, $\mathscr{M}, s', g \vDash \varphi \to \psi$ and $\mathscr{M}, s', g \vDash \varphi$. Therefore, by modus ponens, $\mathscr{M}, s', g \vDash \psi$, and so $\mathscr{M}, s, g \vDash \Box\psi$.

(3) *(Q1)* We want to show that, if x does not occur free in φ, $\vDash \forall x\ (\varphi \to \psi) \to (\varphi \to \forall x\ \psi)$. Choose some arbitrary $\mathscr{M}, s, g$ and let g' be an x-variant of g. Suppose $\mathscr{M}, s, g' \vDash \varphi \to \psi$ and $\mathscr{M}, s, g \vDash \varphi$. By Lemma 3.0.4, notice the second assumption implies that $\mathscr{M}, s, g' \vDash \varphi$. So, by modus ponens, $\mathscr{M}, s, g' \vDash \psi$, i.e. $\mathscr{M}, s, g \vDash \forall x\ \psi$.

(4) *(Q2)* Our goal is to show that $\vDash \forall x\ \varphi \to \varphi\ [a/x]$, for some state symbol a, assuming a is substitutable for x in φ if a is a variable. So suppose $\mathscr{M}, s, g \vDash \forall x\ \varphi$. This means that, if g' is any x-variant of g whatsoever, $\mathscr{M}, s, g' \vDash \varphi$. Then, if a is a state variable, we take $g'(x) = g(a)$. Else, if a is a nominal, we take $g'(x) = V(a)$. In both cases, by applying either Lemma 3.0.5 or Lemma 3.0.6, we arrive at $\mathscr{M}, s, g \vDash \varphi\ [a/x]$.

(5) *(Name)* Next, we prove $\vDash \exists x\, x$. That is, for all $\mathscr{M}, s, g$, there is an x-variant g' of g such that $\mathscr{M}, s, g' \vDash x$. It is clear the required variant exists,

more explicitly:

$$g'(y) = \begin{cases} g(y), & \text{if } y \neq x \\ s, & \text{otherwise} \end{cases}$$

(6) *(Nom)* Let us now prove $\models \forall x\, (\Diamond^n(i \wedge \varphi) \to \Box^m(i \to \varphi))$, for all $n, m \in \mathbb{N}$. Suppose $\mathscr{M}, \mathrm{s}, \mathrm{g}' \models \Diamond^n(i \wedge \varphi)$, for arbitrary $\mathscr{M}, \mathrm{s}, \mathrm{g}$ and g'. We have to show $\mathscr{M}, \mathrm{s}, \mathrm{g}' \models \Box^m(i \to \varphi)$. Using Proposition 2.2.1 and the definition of $\models$, both our assumption and our goal simplify to $\mathscr{M}, \mathrm{V}(i), \mathrm{g}' \models \varphi$.

(7) *(Barcan)* Let $\mathscr{M}, \mathrm{s}, \mathrm{g}$ as usual. We aim to show that $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \forall x\, \Box \varphi$ implies $\mathscr{M}, \mathrm{s}, \mathrm{g} \models \Box \forall x\ \varphi$. It is completely immaterial if we first apply the binder or the modal operator. Both sides of the implication evaluate to $\mathscr{M}, \mathrm{s}', \mathrm{g}' \models \varphi$, for s' and g' with the properties required by the definition of satisfaction.

Now, to show all rules of inference preserve validity:

(8) *(MP)* Suppose $\models \varphi \to \psi$ and $\models \varphi$. By modus ponens in the metalanguage, clearly $\models \psi$.

(9) *(General.)* Suppose $\models \varphi$. That is, all assignment functions g satisfy φ under any model, at any state. So clearly all x-variants of g also satisfy φ, under any model, at any state. That is, $\models \forall x\ \varphi$.

(10) *(Necess.)* Similarly to the last case. Suppose $\models \varphi$. Since φ is true at all states, it must also be true at any state s' accessible from some state s. That is, $\models \Box \varphi$.

$\square$

Theorem 3.0.8 (Soundness)**.** The hybrid logic of $\mathscr{H}(\forall)$ is sound with respect to the class of standard models.

Proof. We have to show $\Gamma \vdash \varphi$ implies $\Gamma \models \varphi$. That is, for any conjunction χ of formulas in Γ, $\vdash \chi \to \varphi$ implies $\models \chi \to \varphi$. This implication is immediate, due to Theorem 3.0.7. $\square$

The consistency of our proof system also follows as a trivial consequence of soundness:

Theorem 3.0.9 (Consistency)**.** For any language $\mathscr{L}(\forall)$, $\nvdash_{\mathscr{L}(\forall)} \bot$.

Proof. Suppose $\vdash \bot$. By Theorem 3.0.7, this implies $\vDash \bot$, which is equivalent to a contradiction in the metalanguage. So $\nvdash \bot$. □

Corollary 3.0.10. The set $\emptyset$ is consistent in any language $\mathscr{L}(\forall)$.

Proof. Immediate, by Proposition 2.3.3.(i). □

Chapter 4

Completeness

In this chapter, we show that the converse of soundness is also true: all hybrid validities are theorems. This property is called completeness. Very soon into this section, we will show that completeness is equivalent to the property that all consistent sets are satisfiable. We will thus devote most of our efforts into constructing a model to satisfy all consistent sets of formulas. Before doing so, however, we will show that it suffices to focus our attention on a particular class of consistent sets. That is, *witnessed maximal consistent sets*. We will begin by proving some statements known as *Lindenbaum's Lemma*, in two different forms, that show that any consistent set is included in some such *maximal* consistent set. We will then introduce a construction called the *canonical model*, and show how it can be refined so as to satisfy all witnessed maximal consistent sets.

This general method for proving completeness is due to Leon Henkin's treatment of classical logic in his seminal paper [19]. The idea of canonical models is specific to modal logic. What we follow here, in additional detail, is Blackburn's and Tzakova's application of these methods to hybrid logic, as can be found in [5]. Our treatment of maximal consistent sets, Lindenbaum's Lemma and canonical models follows as a reference the indications given in [15], [6].

We begin by giving a precise definition of completeness and an equivalent statement of the same property.

Definition 4.0.1 (Completeness)**.** The hybrid logic of $\mathscr{H}(\forall)$ is **complete** with respect to the class of standard models if and only if for all sets of formulas Γ and all formulas φ:

$$\Gamma \models \varphi \text{ implies } \Gamma \vdash \varphi$$

Theorem 4.0.1 (Model Existence)**.** The following statements are equivalent:

(i) The logic of $\mathscr{H}(\forall)$ is complete with respect to the class of standard models;

(ii) For all sets of $\mathscr{L}(\forall)$ -formulas Γ, if Γ is consistent, then Γ is satisfiable.

Proof. (i) $\Longrightarrow$ (ii): Suppose $\mathscr{H}(\forall)$ is complete, i.e. for all Γ, $\Gamma \models \varphi$ implies $\Gamma \vdash \varphi$. Now, suppose by contradiction that there is some Δ such that Δ is consistent and Δ is not satisfiable. By Δ's unsatisfiabilty, we have via Proposition 2.2.3 that $\Delta \models \bot$. Since we assumed completeness, then, $\Delta \vdash \bot$. But this contradicts the assumption that Δ is consistent.

(ii) $\Longrightarrow$ (i): We shall argue by contraposition. Thus, we need to prove that if $\mathscr{H}(\forall)$ is not complete, then there exists some set of formulas Γ such that Γ is consistent and Γ is not satisfiable.

Let, then, Δ and φ be such that $\Delta \models \varphi$ and $\Delta \nvdash \varphi$. From the left part of this assumption, we have $\Delta \cup \{\neg\varphi\} \models \bot$, so by Proposition 2.2.3 it follows that $\Delta \cup \{\neg\varphi\}$ is not satisfiable. Furthermore, from the right part of the assumption and by Proposition 2.3.8, we conclude that $\Delta \cup \{\neg\varphi\}$ is consistent. Thus $\Delta \cup \{\neg\varphi\}$ has the required property. $\square$

So what we are looking for is a satisfying model for each consistent set. In the following subsection, we will show that we can narrow our searches even further, by concentrating only on a small portion of the consistent sets, called (witnessed) maximal consistent sets.

4.1 Lindenbaum's Lemma

Definition 4.1.1 (Maximal Consistent Sets)**.** A set of formulas Γ is **maximal consistent** if and only if:

(i) Γ is consistent

(ii) for all formulas φ, if $\varphi \notin \Gamma$, then $\Gamma \cup \{\varphi\} \vdash \bot$.

In what follows we will abbreviate "maximal consistent set" by "MCS". Below we prove some basic properties of maximal consistent sets which will make them very useful in the proof of completeness for $\mathscr{H}(\forall)$:

Proposition 4.1.1. Let Γ be an MCS. For all formulas φ, ψ, following statements are true:

(1) $\Gamma \vdash \varphi$ iff $\varphi \in \Gamma$

(2) $\varphi \notin \Gamma$ iff $(\neg\varphi) \in \Gamma$

(3) Either $\varphi \in \Gamma$ or $(\neg\varphi) \in \Gamma$.

(4) If $\vdash \varphi$, then $\varphi \in \Gamma$

(5) $\varphi \to \psi \in \Gamma$ iff $\varphi \in \Gamma$ implies $\psi \in \Gamma$

(6) If $\varphi \to \psi \in \Gamma$ and $\varphi \in \Gamma$, then $\psi \in \Gamma$

(7) $\varphi \wedge \psi \in \Gamma$ iff $\varphi \in \Gamma$ and $\psi \in \Gamma$

Proof of (1). Right-to-left is simply the premise rule (Rule i). For the left-to-right direction, suppose (1) $\Gamma \vdash \varphi$ and (2) $\varphi \notin \Gamma$. By the definition of an MCS and deduction, then, (2) becomes $\Gamma \vdash \varphi \to \bot$. Applying modus ponens between (2) and (1) renders $\Gamma \vdash \bot$, which contradicts Γ's consistency. □

Proof of (2). Suppose first $\varphi \notin \Gamma$. By the definition of an MCS and deduction, we have $\Gamma \vdash \varphi \to \bot$, which we can rewrite by the previous equivalence and obtain $(\varphi \to \bot) \in \Gamma$. Recall $\neg\varphi$ is defined as $\varphi \to \bot$. Next, suppose $(\neg\varphi) \in \Gamma$ and $\varphi \in \Gamma$. Rewriting by the previous equivalence, we get $\Gamma \vdash (\neg\varphi)$ and $\Gamma \vdash \varphi$, which contradicts Γ's consistency. □

Proof of (3). Either $\varphi \in \Gamma$ or $\varphi \notin \Gamma$ is true, so we rewrite the right hand side of the disjunction by the previous equivalence. □

Proof of (4). By Proposition 2.3.3.(iii) and equivalence 4.1.1.(1) proved earlier. □

Proof of (5). Left-to-right, after rewriting everything by equivalence 4.1.1.(1), it is a simple case of applying MP (Rule ii).

Right-to-left, suppose $\varphi \in \Gamma$ implies $\psi \in \Gamma$. If $\varphi \in \Gamma$, then by equivalence 4.1.1.(1), we know that (1) $\Gamma \vdash \varphi$ implies $\Gamma \vdash \psi$, and (2) $\Gamma \vdash \varphi$. So we trivially conclude $\Gamma \vdash \psi$, and so, by monotonicity (Prop. 2.3.3.(ii)), $\Gamma \cup \{\varphi\} \vdash \psi$, and, by deduction, $\Gamma \vdash \varphi \to \psi$. Using equivalence 4.1.1.(1) again we obtain $(\varphi \to \psi) \in \Gamma$.

Else, if $\varphi \notin \Gamma$, we know by equivalence 4.1.1.(2) that $(\neg \varphi) \in \Gamma$, so, by equivalence 4.1.1.(1), $\Gamma \vdash \neg \varphi$, and, by deduction, $\Gamma \cup \{\varphi\} \vdash \bot$. We have to show $\Gamma \cup \{\varphi\} \vdash \psi$. By explosion (Tautology 3), this is indeed the case. □

Proof of (6). Immediate via modus ponens, after using the previous equivalence. □

Proof of (7). Trivial consequence of equivalence 4.1.1.(1) and the conjunction introduction / elimination rules (Rules vii and viii). □

For all these properties they have, we are still yet to show that maximum consistent sets exist at all. The answer, however, is that they do. In fact, *any* consistent set of formulas can be extended to an MCS. This is the content of Lindenbaum's Lemma which we are treating in this section.

Lemma 4.1.2 (Existence of Maximal Consistent Sets)**.** There exists a set of $\mathscr{L}(\forall)$ -formulas Δ such that Δ is an MCS.

Proof. Let Γ be any consistent set at all (we know consistent sets exist, by Theorem 3.0.9). Let, furthermore, $\varphi_0, \varphi_1, \ldots$ be an enumeration of all $\mathscr{L}(\forall)$ -formulas. The proof that $\mathscr{L}(\forall)$ is enumerable is beyond the scope of this presentation, but it is essentially the same as in classical logic. We refer the reader to [21] for an explicit proof.

We define a sequence of sets $(\Delta_n)_{n\in\mathbb{N}}$ as follows:

(i) $\Delta_0 = \Gamma$

(ii) $$\Delta_{n+1} = \begin{cases} \Delta_n \cup \{\varphi_n\}, & \text{if } \Delta_n \cup \{\varphi_n\} \nvdash \bot \\ \Delta_n, & \text{otherwise} \end{cases}$$

Notice that the sequence is increasing in the sense of set inclusion; i.e., $\Delta_n \subseteq \Delta_{n+1}$, for all n.

Now, we claim $\Delta = \bigcup\limits_{n \in \mathbb{N}} \Delta_n$ is an MCS. For suppose it were not. That would imply that either $\Delta \vdash \bot$, or there exists some formula ψ such that $\Delta \cup \{\psi\} \nvdash \bot$ and $\psi \notin \Delta$. Let us investigate both sides of this disjunction separately.

Suppose $\Delta \vdash \bot$. By the definition of syntactic consequence, this means that there is a finite subset $\{\chi_1, ..., \chi_n\}$ of Δ such that $\vdash (\chi_1 \wedge ... \wedge \chi_n) \rightarrow \bot$. Now, by the definition of Δ, we know each of $\chi_1, ..., \chi_n$ must belong to *some* set in the sequence. In fact, making use of our observation that that the sequence is increasing, we conclude that there is a set Δ_i that they *all* belong to. So $\{\chi_1, ..., \chi_n\}$ is also a subset of Δ_i for some finite i, and so $\Delta_i \vdash \bot$. Yet that clearly is false. Δ_0 is consistent, for Γ is; and if Δ_n is consistent, then by definition Δ_{n+1} is consistent as well. So $\Delta_i \nvdash \bot$. We conclude that Δ is consistent after all.

Now, could it be the case that there is some formula $\psi \notin \Delta$ such that $\Delta \cup \{\psi\}$ is consistent? Suppose that were true. Since we enumerated all $\mathscr{L}(\forall)$-formulas, we must have encountered ψ as some φ_i in the enumeration. So $\psi = \varphi_i$, for some i. And since $\Delta \cup \{\psi\} \nvdash \bot$, by the contrapositive of monotonicity (Prop. 2.3.3.(ii)) we would have $\Delta_i \cup \{\varphi_i\} \nvdash \bot$. Yet by definition this implies $\varphi_i \in \Delta_{i+1}$, and so $\varphi_i \in \Delta$. We have reached a contradiction.

Therefore, Δ is an MCS. In addition, it is also clear that $\Gamma \subseteq \Delta$. □

The fact that every consistent set is contained in some maximal consistent set follows directly from the previous proof:

Lemma 4.1.3 (Lindenbaum's Lemma)**.** Let Θ be a consistent set of $\mathscr{L}(\forall)$-formulas. Then, there exists a set Δ such that $\Theta \subseteq \Delta$ and Δ is an MCS.

Proof. Construct Δ as in the proof of the previous lemma, letting $\Delta_0 = \Theta$. □

Now, for the purpose of our completeness proof, we will be interested in imposing a stronger restriction on the MCS's that we extend our consistent sets to. That is the property of being *witnessed*:

Definition 4.1.2 (Witnessed Sets)**.** We call a set of formulas Γ **witnessed** if and only if, for every formula φ, if $\exists x\ \varphi \in \Gamma$, then there is some nominal i such that $\exists x\ \varphi \rightarrow \varphi\ [i/x] \in \Gamma$.

To understand why witnessed sets could be useful, we state briefly the contrapositive of Axiom Q2:

Proposition 4.1.4. We have:

(i) $\vdash \varphi\ [i/x] \rightarrow \exists x\ \varphi$

(ii) $\vdash \varphi\ [y/x] \rightarrow \exists x\ \varphi$, for some state variable y substitutable for x in φ

Proof. By contraposition to Axiom Q2. $\square$

It then follows that, if a set Γ is witnessed and maximal consistent, then there is an equivalence between $\exists x\ \varphi \in \Gamma$ and there being a witness i such that $\varphi\ [i/x] \in \Gamma$.

Now, we could ask ourselves, are there any witnessed MCS's at all? The answer is, in fact, yes, granted we have at our disposal enough nominals to satisfy our needs. For, if in some language $\mathscr{L}(\forall)$ we have $\mathrm{NOM} = \emptyset$, then clearly there will not be any witnessed sets definable in $\mathscr{L}(\forall)$. And even if NOM is denumerably infinite, we may not always be able to extend a consistent Γ to a witnessed MCS. As an example, take $\Gamma = \{\exists x\, x\} \cup \{\neg i \mid i \in \mathrm{NOM}\}$. The bottom line is that we would like to have a means of adding new nominals to the signature of a language, ideally in a truth-preserving way. In the expanded language, then, we would be able to construct the desired witnessed MCS. We define all these ideas precisely below.

Definition 4.1.3 (Language Expansions)**.** Let $\mathscr{L}(\forall)$ be a language over PROP, SVAR and NOM, and let $\mathscr{L}^+(\forall)$ be a language over PROP, SVAR and NOM^+. We say $\mathscr{L}^+(\forall)$ is an **expansion** of $\mathscr{L}(\forall)$, written $\mathscr{L}(\forall) \subseteq \mathscr{L}^+(\forall)$, if and only if $\mathrm{NOM} \subseteq \mathrm{NOM}^+$.

Semantically, it is easy to see that language expansions are truth-preserving:

Proposition 4.1.5 (Preservation of Validities)**.** Let $\mathscr{L}(\forall)$ and $\mathscr{L}^+(\forall)$ be two languages such that $\mathscr{L}(\forall) \subseteq \mathscr{L}^+(\forall)$. Then any $\mathscr{L}(\forall)$-formula φ is satisfiable in a model over language $\mathscr{L}(\forall)$ if and only if it is satisfiable in a model over language $\mathscr{L}^+(\forall)$.

Proof. Any model over language $\mathscr{L}(\forall)$ is also a model over language $\mathscr{L}^+(\forall)$, so the left-to-right direction is immediate. For the right-to-left direction, let $\mathscr{M}^+ = \langle \mathrm{W}, \mathrm{R}, \mathrm{V}^+ \rangle$ be a model over $\mathscr{L}^+(\forall)$, let g be an assignment function and $\mathrm{s} \in \mathrm{W}$. Under these assumptions, suppose $\mathscr{M}^+, \mathrm{s}, \mathrm{g} \vDash_{\mathscr{L}^+(\forall)} \varphi$. Take V to be the restriction of V^+ to $\mathscr{L}(\forall)$. Then $\mathscr{M} := \langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$ is clearly a model over $\mathscr{L}(\forall)$. Furthermore, $\mathscr{M}, \mathrm{s}, \mathrm{g} \vDash_{\mathscr{L}(\forall)} \varphi$. $\square$

It gets a little more technical on the syntactic side, but we will show how it can be done. First, we will need to prove the following lemma:

Lemma 4.1.6 (Substituting Nominals)**.** Let φ be a $\mathscr{L}(\forall)$-formula and i a $\mathscr{L}(\forall)$-nominal. Suppose $\vdash \varphi$, and let $\langle \psi_1, ..., \psi_n \rangle$ be its formal derivation. Let x be a state variable that does not occur at all in any ψ_i. Then, the sequence $\langle \psi_1\ [x/i], ..., \psi_n\ [x/i] \rangle$, obtained by substituting x for i in all ψ_i, is a formal derivation of $\varphi\ [x/i]$.

Proof. We will do induction on the length of φ's derivation, n.

If $n = 1$, then φ is an instance of an axiom. We will take the axioms one by one and show that $\varphi\ [x/i]$ is also an instance of some axiom. We will only explicitly treat the following two cases:

1. *(T)* Suppose φ is a tautology and let e be an arbitrary propositional evaluation. We have to show $e(\varphi\ [x/i]) = 1$. Let us define e' as $e'(\psi) = e(\psi\ [x/i])$. It is clear that e' is a propositional evaluation. So, since φ is a tautology, it follows that $e'(\varphi) = 1$. But, by definition, $e'(\varphi) = e(\varphi\ [x/i])$, and thus $e(\varphi\ [x/i]) = 1$.

2. *(Q2 NOM)* Suppose φ is an instance of $\forall y\ \psi \rightarrow \psi\ [j/y]$, for some nominal j and some state variable y. Our wish is to show that $\forall y\ \psi\ [x/i] \rightarrow \psi\ [j/y][x/i]$ is also the instance of an axiom. If $j \neq i$, it is easy to see that $\psi\ [j/y][x/i] = \psi\ [x/i][j/y]$, and $\forall y\ \psi\ [x/i] \rightarrow \psi\ [x/i][j/y]$ is an instance of the NOM version of Q2. Now, consider $j = i$. The key is to notice that $\psi\ [j/y][x/i] = \psi\ [i/y][x/i] = \psi\ [x/i][x/y]$. Since $\forall y\ \psi\ [x/i] \rightarrow \psi\ [x/i][x/y]$ is an instance of the SVAR version Q2, we then reach our desired conclusion.

For axioms K, Name, Nom, Barcan and the SVAR version of Q2, obtaining another instance of the same axioms is a straightforward matter of applying the definition of nominal substitution, so we will not treat them separately.

For the inductive step, $n = m + 1$, φ is either still an axiom, and so this reduces to the base case; φ is obtained by generalization or necessitation on some ψ_j, with $j \leq n$; or φ is obtained by modus ponens on some ψ_j, ψ_k, with $j, k \leq n$. In the case of generalization and necessitation, the induction hypothesis grants us a derivation $\langle \psi_1\ [x/i], ..., \psi_j\ [x/i], ..., \psi_n\ [x/i] \rangle$. By appending $\varphi\ [x/i]$ to this sequence, which by case is the same as either $\forall v\ \psi_j\ [x/i]$ or $\Box\, \psi_j\ [x/i]$, we obtain the desired derivation of $\varphi\,[x/i]$. The case of modus ponens is almost identical, the only difference being that it is a binary rule of inference. $\square$

Now we can give a smooth proof of the following fact:

Proposition 4.1.7 (Preservation of Theorems)**.** Let $\mathscr{L}(\forall)$ and $\mathscr{L}^+(\forall)$ be two languages such that $\mathscr{L}(\forall) \subseteq \mathscr{L}^+(\forall)$. Then, for every $\mathscr{L}(\forall)$-formula φ:

$$\vdash_{\mathscr{L}(\forall)} \varphi \text{ iff } \vdash_{\mathscr{L}^+(\forall)} \varphi$$

Proof. Since $\mathscr{L}^+(\forall)$ contains all instances of the $\mathscr{L}(\forall)$-axioms, and since the rules of inference are the same in $\mathscr{L}^+(\forall)$ and $\mathscr{L}(\forall)$, it follows that any proof in $\mathscr{L}(\forall)$ is also a proof in $\mathscr{L}^+(\forall)$. Thus, the left-to-right direction is evident. In what follows, we will focus on proving the other direction, i.e. showing that $\vdash_{\mathscr{L}^+(\forall)} \varphi$ implies $\vdash_{\mathscr{L}(\forall)} \varphi$.

Let $\langle \psi_1, ..., \psi_n \rangle$ be a derivation of φ in $\mathscr{L}^+(\forall)$. We will do induction on n. If $n = 1$, then φ is an instance of an axiom in $\mathscr{L}^+(\forall)$. And since φ is a $\mathscr{L}(\forall)$-formula, then it is also an instance of an axiom in $\mathscr{L}(\forall)$, so $\vdash_{\mathscr{L}(\forall)} \varphi$.

Now, suppose $n = m + 1$ for some m; and suppose by induction that any $\mathscr{L}(\forall)$-formula ψ that has a derivation of length at most m in $\mathscr{L}^+(\forall)$ has a derivation in $\mathscr{L}(\forall)$. Since φ has a derivation of length $m + 1$, then either φ is an axiom, which reduces to the case for $n = 1$; or one of the following three statements is true:

(i) φ is proved by an application of necessitation. That is, there is a formula ψ_i, with $i \leq n$, such that $\varphi = \Box\,\psi_i$. Now, since φ is a $\mathscr{L}(\forall)$-formula, it follows that ψ_i must be a $\mathscr{L}(\forall)$-formula as well. We thus apply the induction hypothesis and obtain $\vdash_{\mathscr{L}(\forall)} \psi_i$, to which we apply necessitation, yielding $\vdash_{\mathscr{L}(\forall)} \Box\,\psi_i$, which is the same as $\vdash_{\mathscr{L}(\forall)} \varphi$.

(ii) φ is proved by an application of generalization. Similarly as before, we have $\varphi = \forall x\ \psi_i$, so ψ_i is a $\mathscr{L}(\forall)$-formula, so we follow the same reasoning and obtain $\vdash_{\mathscr{L}(\forall)} \varphi$.

(iii) φ is proved by an application of modus ponens. That is, there are $\mathscr{L}^+(\forall)$ formulas ψ_i, ψ_j, with $i, j \leq m$, such that $\psi_j = \psi_i \to \varphi$. If we want to apply the induction hypothesis, we must get rid of all occurrences of $\mathscr{L}^+(\forall)$-nominals in ψ. Thankfully, there is only a finite number of them, since formulas are finite. So we apply Lemma 4.1.6 repeatedly and obtain $\vdash_{\mathscr{L}^+(\forall)} \psi_i'$ and $\vdash_{\mathscr{L}^+(\forall)} \psi_i' \to \varphi$, where ψ_i' denotes the formula in which each occurrence of a $\mathscr{L}^+(\forall)$ -nominal has been replaced by a state variable. Now both ψ_i' and $\psi_i' \to \varphi$ are $\mathscr{L}(\forall)$ -formulas, and their derivations have length less than n. So by the induction hypothesis, $\vdash_{\mathscr{L}(\forall)} \psi_i'$ and $\vdash_{\mathscr{L}(\forall)} \psi_i' \to \varphi$, and by modus ponens, $\vdash_{\mathscr{L}(\forall)} \varphi$.

□

Now that we have richer languages at our disposal, we can prove that witnessed MCS's exist:

Theorem 4.1.8 (Existence of Witnessed Maximal Consistent Sets)**.** Given a language $\mathscr{L}(\forall)$, there exists a set of formulas Δ in a language $\mathscr{L}^+(\forall)$ extended with a denumerably infinite set of nominals, such that Δ is a witnessed MCS.

Proof. Let Γ be a consistent set of $\mathscr{L}(\forall)$ -formulas. Let, furthermore, $\varphi_0, \varphi_1, \ldots$ be an enumeration of all $\mathscr{L}^+(\forall)$ -formulas; and let $i_0, i_1, \ldots$ be an enumeration of all nominals that are in $\mathscr{L}^+(\forall)$ but not in $\mathscr{L}(\forall)$.

We define a sequence of sets $(\Delta_n)_{n\in\mathbb{N}}$ as follows:

(i) $\Delta_0 = \Gamma$

(ii) $$\Delta_{n+1} = \begin{cases} \Delta_n \cup \{\varphi_n\} \cup \{\psi\,[i_n/x]\}, & \text{if } \varphi_n = \exists x\, \psi \text{ and } \Delta_n \cup \{\varphi_n\} \nvdash \bot \\ \Delta_n \cup \{\varphi_n\}, & \text{if } \varphi_n \neq \exists x\, \psi \text{ and } \Delta_n \cup \{\varphi_n\} \nvdash \bot \\ \Delta_n, & \text{otherwise} \end{cases}$$

Let $\Delta = \bigcup_{n\in\mathbb{N}} \Delta_n$. It is clear that Δ is witnessed, since existential formulas are only inserted in Δ along with a witness. The proof that Δ is maximal is identical to the one given under Lemma 4.1.2. In what follows, we will argue that Δ is consistent, so as to finish our proof that Δ is a witnessed MCS.

Following the same reasoning as for Lemma 4.1.2, we conclude that Δ is consistent if each Δ_n is consistent. So first, we notice that Δ_0 is consistent as a set of $\mathscr{L}^+(\forall)$ formulas. Since if it were inconsistent, then, by Proposition 4.1.7, Γ would be inconsistent as a set of $\mathscr{L}(\forall)$ formulas, which contradicts our assumptions.

Now, assume Δ_n is consistent. If $\varphi_n \neq \exists x\, \psi$ or if $\Delta_n \cup \{\varphi_n\} \vdash \bot$, Δ_{n+1} is consistent by definition. The interesting case is the first branch of the inductive step, $\varphi_n = \exists x\ \psi$ and $\Delta_n \cup \{\varphi_n\} \nvdash \bot$. Suppose Δ_{n+1} were inconsistent, hence $\Delta_n \cup \{\varphi_n\} \vdash \neg\psi\ [i_n/x]$. By the definition of syntactic consequence, there exists a finite conjunction χ of formulas in $\Delta_n \cup \{\varphi_n\}$ such that $\vdash \chi \to \neg\psi\ [i_n/x]$. By Lemma 4.1.6 and generalization, we obtain $\vdash \forall y\ (\chi \to \neg\psi\ [y/x])$, for a fresh variable y. Applying Axiom Q1, we get $\vdash \chi \to \forall y\ \neg\psi\ [y/x]$, hence $\Delta_n \cup \{\varphi_n\} \vdash \forall y\ \neg\psi\ [y/x]$. By Theorem 2.3.4, then, we obtain $\Delta_n \cup \{\varphi_n\} \vdash \forall x\ \neg\psi$. But $\varphi_n = \exists x\ \psi$, and so $\Delta_n \cup \{\varphi_n\} \vdash \bot$. This contradicts the assumption that $\Delta_n \cup \{\varphi_n\}$ is consistent.

We thus conclude that Δ is a witnessed MCS in $\mathscr{L}^+(\forall)$. Furthermore, it is clear that $\Gamma \subseteq \Delta$. $\square$

Lemma 4.1.9 (Extended Lindenbaum Lemma)**.** Let Θ be a consistent set of $\mathscr{L}(\forall)$-formulas. Then, there exists a set Δ in a language $\mathscr{L}^+(\forall)$ extended with a denumerably infinite set of nominals, such that $\Theta \subseteq \Delta$ and Δ is a witnessed MCS.

Proof. Construct Δ as in the proof of the previous lemma, letting $\Delta_0 = \Theta$. $\square$

4.2 Completeness via Truth Lemma

We begin by defining precisely what the canonical model is.

Definition 4.2.1 (Canonical Models)**.** The canonical model for $\mathscr{H}(\forall)$ is $\mathscr{M}^C = (W^C, R^C, V^C)$, where:

(i) W^C is the set of all MCS's;

(ii) $\Gamma R^C \Delta$ iff for all formulas φ, $\Box\varphi \in \Gamma$ implies $\varphi \in \Delta$;

(iii) $V^C(a) = \{\Gamma \mid a \in \Gamma \text{ and } \Gamma \text{ is an MCS}\}$, for all propositional symbols or nominals a.

In a similar fashion, we define the canonical assignment.

Definition 4.2.2 (Canonical Assignment)**.** The canonical assignment for $\mathscr{H}(\forall)$ is the function $g^C(x) = \{\Gamma \mid x \in \Gamma \text{ and } \Gamma \text{ is an MCS}\}$, for all variables x.

In what follows, we will need the following properties of canonical models:

Lemma 4.2.1. Let Γ and Δ be two MCS's, and suppose there exists a k-step transition by R^C from Γ to Δ, for some $k > 0$. Then:

(i) If $\Box^k \varphi \in \Gamma$, then $\varphi \in \Delta$.

(ii) If $\varphi \in \Delta$, then $\Diamond^k \varphi \in \Gamma$.

Proof of (i). By induction on k. If $k = 1$, this is implied by the definition of R^C. If $k = n+1$, we have $\Box^n \Box \varphi \in \Gamma$, and we know there is an n-step transition by R^C from Γ to some H, such that $H \mathrm{R}^C \Delta$. Thus, by the induction hypothesis, $\Box \varphi \in H$, and by the definition of R^C, $\varphi \in \Delta$. $\Box$

Proof of (ii). By induction on k. For the base case, we will argue by contradiction. Let $\varphi \in \Delta$, and suppose $\Diamond \varphi \notin \Gamma$. Since Γ is an MCS, we get $\Gamma \vdash \neg \Diamond \varphi$, which, by double negation, is the same as $\Gamma \vdash \Box \neg \varphi$. Now, since we know $\Gamma R^C \Delta$, it follows that $(\neg \varphi) \in \Delta$. This, by Proposition 4.1.1.(2), is equivalent to $\varphi \notin \Delta$. We have reached a contradiction.

Now let $\varphi \in \Delta$ and $k = n+1$. By assumption, we know there is some H such that $HR^C \Delta$, and there is an n-step transition between Γ and H. Making use of the base case, we conclude that $\Diamond \varphi \in H$; and making use of the induction hypothesis, we get $\Diamond^n \Diamond \varphi \in \Gamma$. This is the same as $\Diamond^{n+1} \varphi \in \Gamma$. $\Box$

Lemma 4.2.2 (Existence Lemma for the Canonical Model)**.** Let Γ be an MCS and $\Diamond \varphi \in \Gamma$. Then, there exists some MCS Δ such that $\varphi \in \Delta$ and $\Gamma \mathrm{R}^C \Delta$.

Proof. Let $H = \{\varphi\} \cup \{\psi \mid \Box \psi \in \Gamma\}$. If we can expand H to an MCS, it will be the set we are looking for. We know how to construct such expansions thanks to Lindenbaum's Lemma. But in order to apply it, we have to make sure H is consistent. Let us prove this small fact.

Suppose $H \vdash \bot$. By by the definition of syntactic consequence and the way we defined H, we conclude there is a finite subset $\{\psi_1, ..., \psi_n\}$ of $\{\psi \mid \Box \psi \in \Gamma\}$, such that $\vdash (\psi_1 \wedge ... \wedge \psi_n \wedge \varphi) \to \bot$. By Tautology 13, this implies $\vdash (\psi_1 \wedge ... \wedge \psi_n) \to \neg \varphi$. Applying necessitation, we obtain $\vdash \Box((\psi_1 \wedge ... \wedge \psi_n) \to \neg \varphi)$, and by Axiom K, $\vdash \Box(\psi_1 \wedge ... \wedge \psi_n) \to \Box \neg \varphi$. Thanks to Proposition 2.3.7.(v), we know that $\vdash (\Box \psi_1 \wedge ... \wedge \Box \psi_n) \to \Box(\psi_1 \wedge ... \wedge \Box \psi_n)$. Thus $\vdash (\Box \psi_1 \wedge ... \wedge \Box \psi_n) \to \Box \neg \varphi$. But all $\Box \psi_i$ belong to Γ by definition. So, since Γ is an MCS, using Proposition 4.1.1.(7),(1) we obtain $\Gamma \vdash \Box \neg \varphi$. And since $\Diamond \varphi \in \Gamma$, we have

$\Gamma \vdash \Diamond\varphi$, which is definitionally the same as $\Gamma \vdash \neg\Box\neg\varphi$. We thus reach $\Gamma \vdash \bot$, which contradicts the assumption that Γ is an MCS. So H must be consistent.

Now, we can use Lemma 4.1.3 to obtain an MCS Δ, such that $H \subseteq \Delta$. Thanks to $H \subseteq \Delta$ it follows that, for all formulas ψ, if $\Box\psi \in \Gamma$ then $\psi \in \Delta$. Thus, by definition, $\Gamma R^C \Delta$. Furthermore, we have $\varphi \in \Delta$.

□

Stated as such, notice the canonical model and assignment are not *standard*. For recall from section 2.2 that standard semantics assign exactly one state to each state symbol. Thankfully it is easy to arrive at a standard model if we focus our attention on certain subsets of the canonical model. First, let us introduce the idea of generated submodels.

Definition 4.2.3 (Generated Submodels)**.** Let $\langle \mathrm{W}, \mathrm{R}, \mathrm{V}\rangle$ be a model, g an assignment function, S a subset of W and $\Sigma \in S$. The *generated submodel yielded by* Σ *under* S is $\langle \mathrm{W}^G, \mathrm{R}^G, \mathrm{V}^G\rangle$, where:

(i) $\mathrm{W}^G = \{\Gamma \in S \mid$ there are $k > 0$ and $s_0, \ldots, s_k \in S$ s.t. $s_0 = \Sigma, s_k = \Gamma$ and $s_i \mathrm{R} s_{i+1}$ for all $0 \leq i < k.\}$

(ii) $\Gamma \mathrm{R}^G \Delta$ iff $\Gamma, \Delta \in \mathrm{W}^G$ and $\Gamma \mathrm{R} \Delta$

(iii) $\mathrm{V}^G(a) = \{\Gamma \in \mathrm{W}^G \mid \Gamma \in \mathrm{V}(a)\}$

The assignment function corresponding to the generated submodel is $g^G(x) = \{\Gamma \in \mathrm{W}^G \mid \Gamma \in g(x)\}$.

In particular, we will be working with witnessed models:

Definition 4.2.4. Let $Wit(\mathscr{M}^C)$ be the set of all witnessed MCS's in $\mathscr{M}^C$, and $\Sigma \in Wit(\mathscr{M}^C)$. The *witnessed model* $\mathscr{M}^{\mathscr{W}}$ *yielded by* Σ is the submodel of $\mathscr{M}^C$ generated by Σ under $Wit(\mathscr{M}^C)$.

In other words, witnessed models contain the fraction of the canonical model in which all states are witnessed and accessible from Σ in a finite number of transitions. Since witnessed models are generated submodels, all properties of the latter trickle down to the former. The reason we need witnessed models is because generated submodels of the canonical model have this vital property:

Lemma 4.2.3. Let S be a subset of $\mathscr{M}^C$, $\Sigma \in S$, and $\langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$ the submodel of $\mathscr{M}^C$ yielded by Σ under S. Then, for all nominals i, $\mathrm{V}(i)$ has at most one element.

Proof. We need to show that if $\Gamma \in \mathrm{V}(i)$ and $\Delta \in \mathrm{V}(i)$, then $\Gamma = \Delta$. Since we are working within a generated submodel of the canonical model, we know that Σ, Γ and Δ are all MCS's. Also, we know that there exist $m, n \in \mathbb{N}$ such that Γ and Δ are both accessible from Σ under m and n transitions respectively. Finally, since $\Gamma, \Delta \in \mathrm{V}(i)$, we know by the definition of the canonical model that $i \in \Gamma$ and $i \in \Delta$.

Having established these preliminaries, let us now suppose that $\Gamma \neq \Delta$. This means that there exists some formula φ such that $\varphi \in \Gamma$ and $\varphi \notin \Delta$, or vice-versa. Let us examine the first case; the proof for the second one should obviously proceed in an identical manner. Making use of Proposition 4.1.1.(7) we find that $i \wedge \varphi \in \Gamma$ and (by Proposition 4.1.1.(2)) $i \wedge \neg \varphi \in \Delta$. So, via Lemma 4.2.1 we know that $\Diamond^m(i \wedge \varphi) \in \Sigma$. Now, since all instances of the axioms are instantiated in an MCS (Proposition 4.1.1.(4)), we also know that $\Diamond^m(i \wedge \varphi) \rightarrow \Box^n(i \rightarrow \varphi) \in \Sigma$, so $\Box^n(i \rightarrow \varphi) \in \Sigma$. Since Δ is n transitions away from Σ, by (Lemma 4.2.1), we get that $i \rightarrow \varphi \in \Delta$, so $\varphi \in \Delta$. But we started with the assumption that $\neg \varphi \in \Delta$. We conclude that Γ and Δ must, in fact, be identical. □

Lemma 4.2.4. Let S be a subset of $\mathscr{M}^C$, $\Sigma \in S$, $\langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$ the submodel of $\mathscr{M}^C$ yielded by Σ under S, and g the assignment function corresponding to $\langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$. Then, for all state variables x, $g(x)$ has at most one element.

Proof. Same reasoning as the previous lemma, instantiating Axiom Nom to a state variable instead of a nominal. □

So by making use of Axiom Nom and the properties of MCS's, we find that generated submodels of $\mathscr{M}^C$ assign at most one value to all state symbols. Thus, we can be sure all state symbols in a witnessed model label *almost* uniquely. But some may still not label at all – their valuation / assignment may be the empty set. If that is the case, we need to glue on a new state to our witnessed model, under which ensure all previously unassigned state symbols are true. We name this construction the *completed model*.

Definition 4.2.5 (Completed Model)**.** Let $\langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$ be the witnessed model generated by some witnessed MCS Σ and g the corresponding assignment. If all state symbols are assigned by V or g to some $s \in W$, then the completed model of Σ is $\langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$ itself, and the completed assignment is g.

Otherwise, the completed model is $\langle \mathrm{W}^s, \mathrm{R}^s, \mathrm{V}^s \rangle$, where:

(i) $\mathrm{W}^s = \mathrm{W} \cup \{\star\}$, where $\star$ is *not* an MCS. (Say, $\star = \{\bot\}$.)

(ii) $\mathrm{R}^s = \mathrm{R} \cup (\star, \Sigma)$

(iii) $\mathrm{V}^s(i) = \begin{cases} \{\star\}, & \text{if } \mathrm{V}(i) = \emptyset \\ \mathrm{V}(i), & \text{otherwise} \end{cases}$

And the completed assignment is g^s, where $g^s(x) = \begin{cases} \{\star\}, & \text{if } g(x) = \emptyset \\ \mathrm{g}(x), & \text{otherwise} \end{cases}$

Now we have finally constructed a standard model based on the canonical model. They are precisely the models we need in order to apply the model existence theorem. It is very important to note that we do not attach the state $\star$ by default. We only attach it if we need it. As a result, we can be sure that all states in the completed model satisfy some state symbol, even if that state is $\star$. Here is a small proof, to make this idea clear.

Lemma 4.2.5. Let s be a state in the completed model of Σ. Then there exists a nominal i such that $s \in V(i)$, or there exists a state variable x such that $s \in g(x)$.

Proof. The state s is either a witnessed MCS or the dummy state $\star$. If s is a witnessed MCS, then it contains all instances of the axioms, including $\exists x\, x$ for some state variable x. Since it is witnessed, then it must also contain $x\,[i/x]$ for some nominal i, which is obviously identical to i. Thus, $i \in s$ and so, by definition, $s \in V(i)$.

Now suppose $s = \star$, and let V^w, g^w be the valuation and the assignment function of the witnessed model generated by Σ. Since $s = \star$, it follows by the definition of the completed model that there is either a nominal i such that $\mathrm{V}^w(x) = \emptyset$, or there is a variable x such that $\mathrm{g}^w(x) = \emptyset$. Hence, it is the case that $\mathrm{V}(i) = \star$, or $\mathrm{g}(i) = \star$. $\square$

Lemma 4.2.6. Let Δ be a witnessed MCS and $\Diamond\varphi \in \Delta$. Then, for all formulas ψ and variables x, there is some nominal i such that $\Diamond((\exists x\, \psi \to \psi\,[i/x]) \wedge \varphi) \in \Delta$.

Proof. Since Δ is closed under modus ponens (Proposition 4.1.1.(6)) and Δ is witnessed, the result is immediate from Proposition 2.3.7.(iv). $\square$

Intuitively, this lemma takes as "input" a diamond-formula ($\Diamond\varphi$), and returns as "output" another diamond formula ($\Diamond((\exists x\ \psi \to \psi\ [i/x]) \wedge \varphi)$). This makes it well suited for recursive applications over its own self. This is the idea used to prove the existence lemma:

Lemma 4.2.7 (Existence Lemma for Completed Models)**.** Let $\mathscr{M} = \langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$ be a completed model and Δ be a witnessed MCS in W. If $\Diamond\varphi \in \Delta$, then there exists a witnessed MCS Γ in W such that $\varphi \in \Gamma$ and $\Delta R \Gamma$.

Proof. Let $\exists x_1\ \psi_1, \exists x_2\ \psi_2, \ldots$ be an enumeration of all existentially quantified $\mathscr{L}(\forall)$ -formulas. We start by defining a family of sets:

- $\Gamma_0 \quad := \{\varphi\} \cup \{\psi \mid \Box\psi \in \Delta\}$
- $\Gamma_{n+1} := \Gamma_n \cup \{\exists v\ \chi \to \chi\ [j/v]\}$, where $v = x_{n+1}, \chi = \psi_{n+1}$, and $j = i_{n+1}$, the $n+1$-th term in a sequence of nominals $(i_n)_{n\in\mathbb{N}}$.

This family, of course, is dependent on the choice of nominals. Our claim is that it is possible to choose a suitable sequence $(i_n)_{n\in\mathbb{N}}$ such that, for all $n \in \mathbb{N}$:

(a.) Γ_n is consistent, and

(b.) $\Diamond((\exists x_1\ \psi_1 \to \psi_1\ [i_1/x_1]) \wedge \ldots \wedge (\exists x_n\ \psi_n \to \psi_n\ [i_n/x_n]) \wedge \varphi) \in \Delta$

First, by an identical argument as the one used in Lemma 4.2.2, we note that Γ_0 is consistent. Condition $(b.)$ also holds for Γ_0, since the left hand side of the conjunction is empty and $\Diamond\varphi \in \Delta$.

Next, assume the required properties are satisfied at step n. We will show how to choose i_{n+1} given the choice of $i_0, \ldots, i_n$ so that the two properties hold for $n+1$ as well. With the help of Lemma 4.2.6, it is straightforward:

(1.) $\Diamond((\exists x_1\ \psi_1 \to \psi_1\ [i_1/x_1]) \wedge \ldots \wedge (\exists x_1\ \psi_n \to \psi_n\ [i_n/x_n]) \wedge \varphi) \in \Delta$ *(by i.h.)*

$\Longrightarrow$ *(by Lemma 4.2.6 applied to last line, ψ_{n+1} and x_{n+1})* there is some nominal j s.t.:

(2.) $\Diamond((\exists x_1\ \psi_1 \to \psi_1\ [i_1/x_1]) \wedge \ldots \wedge (\exists x_{n+1}\ \psi_{n+1} \to \psi_{n+1}\ [j/x_{n+1}]) \wedge \varphi) \in \Delta$

So we pick a j satisfying (2.) as our i_{n+1}, and we let $\Gamma_{n+1} = \Gamma_n \cup \{\exists x_{n+1}\ \psi_{n+1} \to \psi_{n+1}\ [i_{n+1}/x_{n+1}]\}$. At step $n+1$, condition (b.) holds. Let us then check that Γ_{n+1} is consistent.

Suppose $\Gamma_{n+1} \vdash \bot$. Thanks to the way we defined Γ_{n+1}, that means we have $\vdash \tau \to \neg(w_1 \wedge ... \wedge w_{n+1} \wedge \varphi)$, where τ is a conjunction of formulas in $\{\psi \mid \Box\psi \in \Delta\}$, and w_j is an abbreviation for $(\exists x_j\ \psi_j \to \psi_j\ [i_j/x_j])$. Generalizing and applying axiom K, we obtain $\vdash \Box\tau \to \neg\Diamond(w_1 \wedge ... \wedge w_{n+1} \wedge \varphi)$. But $\Box\tau \in \Delta$, so $\neg\Diamond(w_1 \wedge ... \wedge w_{n+1} \wedge \varphi) \in \Delta$, which, since we know (2.) is true, contradicts Δ's consistency. So Γ_{n+1} is, in fact, consistent.

To finish off the proof, let $\Gamma^+ = \bigcup_{n\in\mathbb{N}} \Gamma_n$. By construction, Γ^+ is witnessed. And, since all Γ_n are consistent, Γ^+ is consistent as well. We apply the regular version of Lindenbaum's Lemma (4.1.3) to Γ^+ and obtain a set Γ which is witnessed, maximal and consistent, and contains ψ for all $\Box\psi \in \Delta$. So $\Delta \mathrm{R} \Gamma$, and $\varphi \in \Gamma$. $\Box$

And now, the moment of Truth:

Lemma 4.2.8 (Truth Lemma)**.** Let $\mathscr{M} = \langle \mathrm{W}, \mathrm{R}, \mathrm{V} \rangle$ be a completed model, g its assignment function, and Δ a witnessed MCS in W. Then, for all formulas φ,

$$\varphi \in \Delta \text{ iff } \mathscr{M}, \Delta, g \vDash \varphi$$

Proof. We proceed by induction on φ. If $\varphi = \bot$, both the left-to-right and right-to-left assumptions entail contradictions, so the equivalence holds ex falso. If φ is a nominal, propositional variable or state variable, the equivalence holds by the definition of completed models and completed assignments.

Suppose now we have $\varphi \in \Delta$ iff $\mathscr{M}, \Delta, g \vDash \varphi$ and $\psi \in \Delta$ iff $\mathscr{M}, \Delta, g \vDash \psi$. We want to show $(\varphi \to \psi) \in \Delta$ iff $\mathscr{M}, \Delta, g \vDash (\varphi \to \psi)$. Notice that, by Proposition 4.1.1.(5) and the definition of satisfaction, this is equivalent to showing that ($\varphi \in \Delta$ implies $\psi \in \Delta$) iff ($\mathscr{M}, \Delta, g \vDash \varphi$ implies $\mathscr{M}, \Delta, g \vDash \psi$). So rewrite either side by the induction hypothesis to arrive at the desired conclusion.

For $\Diamond\varphi$, the left-to-right direction is precisely the existence lemma proved earlier. So suppose, for the other direction, that $\mathscr{M}, \Delta, g \vDash \Diamond\varphi$, meaning that there is some state s' in the completed model such that $\Delta \mathrm{R} \mathrm{s}'$ and $\mathscr{M}, s', g \vDash \varphi$. By the induction hypothesis, we know that if s' is a witnessed MCS, then $\varphi \in s'$. But s' could also be $\star$. Now, since $\Delta \mathrm{R} \mathrm{s}'$, we know $\mathrm{s}' \neq \star$ (since $\star$ was defined as having no transitions leading into it), and so we can safely conclude $\varphi \in \mathrm{s}'$. Now we use Lemma 4.2.1 and get $\Diamond\varphi \in \Delta$ to complete the proof.

Lastly, we consider the case for $\exists x\ \varphi$. Assume, first, that $\exists x\ \varphi \in \Delta$. Since

Δ is a witnessed MCS, then, $\varphi\,[i/x] \in \Delta$ for some i. By applying the induction hypothesis to $\varphi\,[i/x]$ we get $\mathscr{M},\Delta,g \models \varphi\,[i/x]$. Now we use (weak) Soundness (Theorem 3.0.7) and Proposition 4.1.4 to get $\models (\varphi\,[i/x] \to \exists x\ \varphi)$, which implies $\mathscr{M},\Delta,g \models \exists x\ \varphi$.

For the reverse direction, assume there exists an x-variant g$'$ of g such that $\mathscr{M},\Delta,g' \models \varphi$. Now take a look at s $=$ g$'(x)$, the state that g$'$ assigns to variable x. Since s is a state in the completed model, recall from Lemma 4.2.5 that either some nominal is true at s under V, or some state variable is true at s under g. Let us examine both parts of this disjunction.

If V$(i) =$ g$'(x)$ for some i, and since g$'$ is an x-variant of g, we apply Lemma 3.0.6 and find that $\mathscr{M},\Delta,g \models \varphi\,[i/x]$. Once again, we use the induction hypothesis on $\varphi\,[i/x]$ and conclude that $\varphi\,[i/x] \in \Delta$. So, by Proposition 4.1.4, $\exists x\ \varphi \in \Delta$.

Now, if g$(y) =$ g$'(x)$ for some y, we need some more preparations. We cannot apply Lemma 3.0.5 directly, since y might not be substitutable for x in φ. So let φ' be the y-free variant of φ. By Soundness and Corollary 2.3.9, we know that $\models (\varphi \leftrightarrow \varphi')$, so $\mathscr{M},\Delta,g' \models \varphi'$. Since y is now substitutable for x in φ', we finally apply Lemma 3.0.5 and obtain $\mathscr{M},\Delta,g \models \varphi'\,[y/x]$. Hence, by induction $\varphi'\,[y/x] \in \Delta$; hence, by Proposition 4.1.4, $\exists x\ \varphi' \in \Delta$. From Corollary 2.3.9 it also easily follows that $\vdash (\exists x\ \varphi \leftrightarrow \exists x\ \varphi')$. Since Δ is an MCS, we use this equivalence to conclude that $\exists x\ \varphi \in \Delta$. □

Theorem 4.2.9 (Completeness)**.** The hybrid logic of $\mathscr{H}(\forall)$ is complete with respect to the class of standard models.

Proof. By Theorem 4.0.1, the statement we have to prove is equivalent to showing that any consistent set Γ of $\mathscr{L}(\forall)$ -formulas is satisfiable. Suppose, therefore, that Γ is consistent. We use Lemma 4.1.9 to extend Γ to a witnessed MCS Γ^+ in an extended language $\mathscr{L}^+(\forall)$. Let $\mathscr{M}$ be the completed model generated by Γ^+, and g its assignment function. Then, by Lemma 4.2.8, we have that $\mathscr{M},\Gamma^+,\mathrm{g} \models_{\mathscr{L}^+(\forall)} \Gamma^+$, and since $\Gamma \subseteq \Gamma^+$, we get $\mathscr{M},\Gamma^+,g \models_{\mathscr{L}^+(\forall)} \Gamma$. So Γ is satisfiable in $\mathscr{L}^+(\forall)$, which, by Proposition 4.1.7, is equivalent to Γ being satisfiable in $\mathscr{L}(\forall)$. □

Chapter 5

Formalizing Hybrid Logic in Lean

5.1 Brief Overview of the Language

The Lean project was started by Leonardo de Moura in 2013 [33]. It has since gone through a number of major releases, the latest being Lean 4, which is still in development stage. Its aim is to provide mathematicians with a way to express formal statements, and to write proofs which are computer checked for correctness. Thanks to the community driven project Mathlib, considerable areas of mathematics have been formalized in Lean and are available via a library import to anyone who intends to do computer-verified formal reasoning. Freek Wiedijk of Radboud University of Nijmegen maintains a list of 100 mathematical theorems and their formalizations in different proof assistants, as a means of comparing the effective proving power of different systems [40]. As of August 2023, 76 of these have been formalized in Lean, including, for example, The Independence of the Continuum Hypothesis [17]. In this list, Lean is only surpassed by Isabelle, HOL and Coq, all of which are programming languages with a much longer history and a much larger community.

Lean is based on a version of type theory known as the Calculus of Inductive Constructions [1]. Lean defines an infinite hierarchy of types, called `Sorts`. The bottom-most level in this hierarchy is the one which will interest us the most. It is inhabited by *propositions*, which determine the type `Prop`. Examples of elements of type `Prop` include `0 = 0` or `True`. We call the elements of a type its *terms*. The way in which Lean can assist in theorem-proving is thanks to the Curry-Howard isomorphism [20], which establishes a correspondence between proofs in constructive logic and computer programs in typed programming languages. Under this correspondence, we treat each proposition `p : Prop` as a type in its own self; namely, the type of its *proofs*. The relation between a term `p : Prop` and a term `h : p` is thus the same as that between a logical statement and its proof. Props are statements; the mere mention of a statement is not a mark of its truth. The terms of Props are proofs; their existence is what validates a given sentence. Take the proposition `True : Prop`. If we write proofs, we might expect to always have a proof of `True` on hand. And indeed, in Lean we have `trivial : True`, i.e., `trivial` is a term of type `True`.

Thus, in the proposition-as-types paradigm, theorem proving becomes a matter of constructing terms of appropriate types. Given a proposition `p : Prop`, we prove `p` by constructing a term `pf : p`. The methods used in the construction of such terms then correspond to the rules in a natural-deduction type proof system. To understand how this is possible, we provide the example of the Or connector. Internally (in the file Prelude.lean), Lean defines all logical connectors as inductive types. This equates to giving an inductive specification of what it means to prove a proposition containing the respective propositional connector. In particular:

```
inductive Or (a b : Prop) : Prop where
  | inl (h : a) : Or a b
  | inr (h : b) : Or a b
```

This definition allows for creating terms of type `Or a b` through any of two different constructors: `Or.inl`, which takes a proof of `a` and returns a proof of `Or a b`; or `Or.inr`, which takes a proof of `b` and returns a proof of `Or a b`. This is the same way in which we construct proofs of disjunctive propositions in logic: we conclude $a \vee b$ if either a is true, or b is true. Accordingly, Lean offers a way to eliminate propositions into other propositions, as we are able to in natural deduction as well. For take the disjunction elimination rule: if $a \vee b$, and $a \rightarrow c$ and $b \rightarrow c$, then c. In Lean, this is obtained through pattern matching:

```
theorem Or.elim {c : Prop} (h : Or a b) (left : a → c) (right : b
    → c) : c :=
  match h with
  | Or.inl h => left h
  | Or.inr h => right h
```

Here, the Lean interpreter checks the term `h : Or a b` against all the constructors that could be used to create it; in effect "destructing" `h` back into a proof of `a` or, respectively, a proof of `b`. In both cases, a term of type `c` can be constructed, by an application of either `left` or `right`.

In fact, this last example also illustrates how we deal with implication in Lean. A proof of `p → q` (implication introduction) is a *function* from type `p` to type `q`. This is also called *function abstraction*. Implication elimination, or modus ponens, is simply a matter of *function application*: given a term `f : p → q` and a term `h : p`, we obtain a term of type `q` simply by applying the function, `f h`. In our previous example, we applied the functions `left` and `right` to obtain the desired proof of `c`.

All other propositional connectives then behave in the expected manner. Conjunction has one constructor (such inductive types are called *structures* in Lean), `And.intro p q`. It produces a proof of `p ∧ q` given a proof `p` and a proof of `q`. It has two elimination rules, `And.left` and `And.right`, wihch return either `p` or `q` respectively, given a proof of `p ∧ q`. Equivalence, `p ↔ q`, is introduced by `Iff.intro` from a proof of `p → q` and a proof of `q → p`, and it eliminates via `Iff.mp` or `Iff.mpr` back to `p → q` or `q → p` respectively. Negation, `¬p`, is defined as `p → False`, so the rules governing its use are function abstraction and application.

First-order reasoning is provided thanks to a feature of Lean called dependent typing. Lean implements the notions of Π-types and Σ-types. Π-types are the type of dependent functions: functions in which the type returned depends on the type that they are applied to. Under the Howard-Curry isomorphism, they represent the counterpart to constructive proofs of universal formulas. Constructively, given a domain A and a predicate p, to prove $\forall x\, p(x)$ is to define a function which maps any element $x \in A$ to a proof of $p(x)$. In Lean, given a type α and a term `p : α → Prop`, we define a function from `x : α` to `p x`. Intuitively, this function proves the statement `p x` for every term `x : α` it is provided; accordingly, its output type (the statement it proves) depends on `x`. And in fact, Lean treats regular functions as special cases of dependent functions. So the rules governing universal quantification as the same as those for implication: (dependent) function abstraction, which we have just explained, and (dependent) function application.

Σ-types can be viewed as the type of dependent cartesian products: the type of pairs $\langle a, b\rangle$ where b's type may depend on a. Now, to prove existential statements constructively is to provide just such an object: a proof of $\exists x\, p(x)$ is a pair $\langle x, p(x)\rangle$, where x is some element in the domain, and $p(x)$ is a proof that p holds for x. So specifically, what we need is a type of dependent products in which the second element is a proof of a `Prop`. In Lean, these are defined simply as follows:

```
inductive Exists {α : Sort u} (p : α → Prop) : Prop where
  | intro (w : α) (h : p w) : Exists p
```

The syntax `Σ (x : α), p x` exists as well, but it is reserved for higher Sorts. Lean features something known as *proof-irrelevance*, which states that any two proofs of a given `Prop` are equal. The details of proofs are lost at compile-time. This means that, in general, we will not be able to extract the witness used to prove an existential statement out of its proof, since `Exists` is defined as an inductive `Prop`. If we are proving another `Prop`, however, Lean allows us to do pattern-matching on an `Exists`, in effect ensuring that an existential-elimination rule is provided. This is not a concern for dependent products of types other than `Prop`, in which `Σ (x : α), p x` is used.

With introduction and elimination rules for all first-order connectives, proofs in Lean can then be given by a composition of rule applications, yielding a term of the desired type. This roughly corresponds to what is known as *term mode* proofs. Lean, however, also offers the functionality to construct bottom-up proofs, by determining the current *goals* of the proof and providing backwards-reasoning towards those goals. These are known as *tactic mode* proofs. For example, if the current goal is:

```
⊢ ∀  (Γ : Set Form), Set.consistent Γ → ∃  Γ', Γ ⊆ Γ' ∧
    Set.MCS Γ'}
```

(which is the statement of Lindenbaum's Lemma in our formalization), we may use the tactic `intro Γ cons`. This corresponds to two applications of function abstraction; or, more intuitively, to writing "Let Γ be a set. Suppose Γ is consistent" in a mathematical proof. Lean would then add `Γ: Set Form` and `cons: Set.consistent Γ` as terms in the current context and then rewrite the goal as:

```
⊢ ∃  Γ', Γ ⊆ Γ' ∧ Set.MCS Γ'
```

This new goal, in turn, could be proved by an application of the `exists` tactic. We direct the reader to [38] for a reference on all core Lean 4 tactics.

This concludes our short introduction to the Lean programming language. A more thorough description is available in [1] and [39]. It now becomes our task to embrace the role of translators and show how a proof given in the metalanguage of set theory (namely, the proof of hybrid completeness, Theorem 4.2.9) can be transposed to Lean's type theory.

But before advancing to the next section, we make one last remark. We mentioned Lean's proving power correponds to that of constructive logic. We will however not concern ourselves with staying within the limits of constructive logic in our formalization. We will use the `Classical` library, which provides us with the law of excluded middle and the Axiom of Choice. We will also use `Mathlib`, mainly in order to deal with sets and their properties.

5.2 The Implementation

Here we detail our main design decisions; some difficulties we faced; our proposed solutions; and challenges still ahead. Our full code is available at [34].

5.2.1 Main Definitions

We define the propositional variables, state variables and nominals as records around $\mathbb{N}$:

```
structure PROP where
  letter : Nat
deriving DecidableEq, Repr

structure SVAR where
  letter : Nat
deriving DecidableEq, Repr

structure NOM where
  letter : Nat
deriving DecidableEq, Repr
```

Using these definitions, we define the inductive type Form of well-formed formulas thus:

```
inductive Form where
  | bttm : Form
  | prop : PROP → Form
  | svar : SVAR → Form
  | nom  :  NOM → Form
  | impl : Form → Form → Form
  | box  : Form → Form
  | bind : SVAR → Form → Form
deriving DecidableEq, Repr
```

And we define the other connectives as abbreviations of longer formulas:

```
def Form.neg      : Form → Form := λ φ => Form.impl φ Form.bttm
def Form.conj     : Form → Form → Form := λ φ => λ ψ =>
   Form.neg (Form.impl φ (Form.neg ψ))
def Form.iff      : Form → Form → Form := λ φ => λ ψ =>
   Form.conj (Form.impl φ ψ) (Form.impl ψ φ)
def Form.disj     : Form → Form → Form := λ φ => λ ψ =>
   Form.impl (Form.neg φ) ψ
def Form.diamond  : Form → Form := λ φ => Form.neg (Form.box
   (Form.neg φ))
def Form.bind_dual: SVAR → Form → Form := λ x => λ φ =>
   Form.neg (Form.bind x (Form.neg φ))
```

In the rest of the file, we give the necessary definitions for occurrences of variables (free or otherwise) and substitutions. All these definitions are essentially the same as the natural-language definitions given in section 2.1.

```
def occurs (x : SVAR) (φ : Form) : Bool :=
  match φ with
  | Form.bttm     => false
  | Form.prop _   => false
  | Form.svar y   => x = y
  | Form.nom  _   => false
  | Form.impl φ ψ => (occurs x φ) || (occurs x ψ)
  | Form.box  φ   => occurs x φ
  | Form.bind _ φ => occurs x φ
```

```
def is_free (x : SVAR) (φ : Form) : Bool :=
  match φ with
  | Form.bttm     => false
  | Form.prop _   => false
  | Form.svar y   => x == y
  | Form.nom  _   => false
  | Form.impl φ ψ => (is_free x φ) || (is_free x ψ)
  | Form.box  φ    => is_free x φ
  | Form.bind y φ => (y != x) && (is_free x φ)

def is_bound (x : SVAR) (φ : Form) := (occurs x φ) && !(is_free x
    φ)

def subst_svar (φ : Form) (s : SVAR) (x : SVAR) : Form :=
  match φ with
  | Form.bttm     => φ
  | Form.prop _   => φ
  | Form.svar y   => ite (x = y) s y
  | Form.nom  _   => φ
  | Form.impl φ ψ => (subst_svar φ s x) → (subst_svar ψ s x)
  | Form.box  φ    => □ (subst_svar φ s x)
  | Form.bind y φ => ite (x = y) (Form.bind y φ) (Form.bind y
    (subst_svar φ s x))

def is_substable (φ : Form) (y : SVAR) (x : SVAR) : Bool :=
  match φ with
  | Form.bttm     => true
  | Form.prop _   => true
  | Form.svar _   => true
  | Form.nom  _   => true
  | Form.impl φ ψ => (is_substable φ y x) && (is_substable ψ y x)
  | Form.box  φ    => is_substable φ y x
  | Form.bind z φ =>
      if (is_free x φ == false) then true
      else z != y && is_substable φ y x
```

We also define a way to construct formulas with iterated modal operators, as necessary, for example, for Axiom Nom.

```
def iterate_nec (n : Nat) (φ : Form) : Form :=
  let rec loop : Nat → Form → Form
    | 0, φ    => φ
    | n+1, φ => □ (loop n φ)
  loop n φ

def iterate_pos (n : Nat) (φ : Form) : Form :=
  let rec loop : Nat → Form → Form
    | 0, φ    => φ
    | n+1, φ => ◊ (loop n φ)
  loop n φ
```

In the file Truth.lean, we introduce a semantics for our language. First, we define models thus:

```
structure Model where
  W : Type
  R : W → W → Prop
  Vₚ: PROP → Set W
  Vₙ: NOM  → W
```

Defined set-theoretically, we saw that models require a set of states. The Lean-equivalent to this set of states is the field W in the structure Model. We let W be any type, thus the states of our models being terms of type W. Accordingly, we define the accessibility relation R as a binary relation over terms of type W. Recall that, in section 2.2, we provided our models with only one valuation function, responsible for evaluating both propositional variables and nominals. In Lean, since we have defined PROP and NOM to be separate types, we accordingly define two different valuations: V_p, which is a function from PROP to Set W (i.e., sets of states); and V_n, which is a function from NOM to W. Similarly, assignment functions are defined as functions from SVAR to W:

```
def I (W : Type) := SVAR → W
```

Notice that, by these definitions of V_n and I, our models and assignment functions are by default *standard*. This was a conscious design decision, in order to simplify our semantic reasoning. It is true that, in section 4.2, we had to briefly work with nonstandard models. For this reason, we will see later that we also introduced the notion of *general models* in Lean, in which the valuation of nominals and the assignment function have *Set W* as a codomain, instead of W.

Building upon these definitions, the satisfaction relation is then defined as follows:

```
def is_variant (g₁ g₂ : I W) (x : SVAR) := ∀  y : SVAR, ((x ≠ y)
    → (g₁ y = g₂ y))

def Sat (M : Model) (s : M.W) (g : I M.W) : (φ : Form) → Prop
  | Form.bttm      => False
  | Form.prop p    => s ∈ (M.Vₚ p)
  | Form.nom  i    => s = (M.Vₙ i)
  | Form.svar x    => s = (g x)
  | Form.impl ψ χ => (Sat M s g ψ) → (Sat M s g χ)
  | Form.box  ψ    => ∀  s' : M.W, (M.R s s' → (Sat M s' g ψ))
  | Form.bind x ψ => ∀  g' : I M.W, ((is_variant g' g x) → Sat M
    s g' ψ)
```

This mirrors the definition we gave under section 2.2. Validity, set satisfaction, entailment and satisfiability are also defined in the expected manner. At this point, we are already able to do some small proofs, mostly automated by Lean's simplifier. This is how we proved that the additional connectives are interpreted in the standard way by our satisfaction relation:

```
theorem neg_sat : ((M,s,g) ⊨ ∼φ) ↔ ((M,s,g) ⊭ φ) := by
  simp only [Sat, or_false]
theorem and_sat : ((M,s,g) ⊨ φ ⋀ ψ) ↔ (((M,s,g) ⊨ φ) ∧ (M,s,g)
    ⊨ ψ) := by simp
theorem or_sat  : ((M,s,g) ⊨ φ ⋁ ψ) ↔ (((M,s,g) ⊨ φ) ∨ (M,s,g)
    ⊨ ψ) := by simp
theorem pos_sat : (((M,s,g) ⊨ ◊φ) ↔ (∃  s' : M.W, (M.R s s' ∧
    (M,s',g) ⊨ φ))) := by simp
theorem ex_sat  : ((M,s,g) ⊨ ex x, φ) ↔ (∃  g' : I M.W,
    (is_variant g' g x) ∧ ((M,s,g') ⊨ φ)) := by
  simp [-is_variant]
theorem iff_sat : ((M,s,g) ⊨ (φ ↔ ψ)) ↔ (((M,s,g) ⊨ φ) ↔
    (M,s,g) ⊨ ψ) := by
  rw [Form.iff, and_sat, Sat, Sat]
  apply Iff.intro
  . intro ⟨h1, h2⟩
    apply Iff.intro <;> assumption
  . intro h1
    apply And.intro <;> simp [h1]
```

In the file Tautology.lean we define propositional evaluations, tautologies and we prove the tautologies we will use in formal derivations. The required definitions are listed below.

```
structure Eval where
  f  : Form → Bool
  p1 : f ⊥ = false
  p2 : ∀  φ ψ : Form, (f (φ → ψ) = true) ↔ (¬(f φ) = true ∨
    (f ψ) = true)

def Tautology (φ : Form) : Prop := ∀  e : Eval, e.f φ = true
```

This allows us to define the proof system inductively as:

```
inductive Proof : Form → Prop where
  | general {φ : Form} (v : SVAR):
        Proof φ → Proof (all v, φ)
  | necess {φ : Form}:
        Proof φ → Proof (□ φ)
  | mp {φ ψ : Form}:
        Proof (φ → ψ) → Proof φ → Proof ψ
  | tautology {φ : Form}:
        Tautology φ → Proof φ
  | ax_k {φ ψ : Form}:
        Proof (□ (φ → ψ) → (□ φ → □ ψ))
  | ax_q1 (φ ψ : Form) {v : SVAR} (p : is_free v φ = false):
        Proof ((all v, φ → ψ) → (φ → all v, ψ))
  | ax_q2_svar (φ : Form) (v s : SVAR) (p : is_substable φ s v):
      Proof ((all v, φ) → φ[s // v])
  | ax_q2_nom (φ : Form) (v : SVAR) (s : NOM):
      Proof ((all v, φ) → φ[s // v])
  | ax_name (v : SVAR):
      Proof (ex v, v)
  | ax_nom {φ : Form} {v : SVAR} (m n : Nat):
    Proof (all v, (iterate_pos m (v ⋀ φ) → iterate_nec n (v →
  φ)))
  | ax_brcn {φ : Form} {v : SVAR}:
    Proof ((all v, □ φ) → (□ all v, φ))
```

The next step is to define the notion of syntactic consequence: $\Gamma \vdash \varphi$ if and only if there exists a conjunction of a finite subset of formulas in Γ such that $\vdash \mathit{conjunction}\, \Gamma \to \varphi$. The basis of our implementation is inspired by the formalization of Public Announcement Logic in Lean 3 [23], where the conjunction of a List of formulas is defined thus:

```
def conjunction {α agent : Type} :
  list (sentence α agent) → sentence α agent
| []                := ⊥ ↣ ⊥
| [φ]               := φ
| (φ :: Γ)          := φ & conjunction Γ
```

However, we note that Lists in Lean are finite in length [31]. We shall wish to have a definition of conjunction which works for sets of arbitrary cardinality, including infinite. Here, we make use of Lean's subtyping capabilities. Give a term Γ of type `Set Form`, it is easy to define a type akin to that of *finite subsets of* Γ. That type is `List Γ`; it is equivalent to `List {φ : Form // Γ φ}`. With this on hand, we give the following definitions to conjunctions and syntactic consequence:

```
def conjunction (Γ : Set Form) (L : List Γ) : Form :=
match L with
  | []      => ⊥ → ⊥
  | h :: t => h.val ⋀ conjunction Γ t

def SyntacticConsequence (Γ : Set Form) (φ : Form) : Prop := ∃
    L, Proof ((conjunction Γ L) → φ)
```

Our definitions of consistency, maximal consistency and witnessed sets then mirror the definitions given in sections 2.3 and 4.1:

```
def Set.consistent (Γ : Set Form) := Γ ⊬ ⊥

def Set.MCS (Γ : Set Form) := Γ.consistent ∧ (∀  {φ : Form},
    (¬φ ∈ Γ) → ((Γ ∪ {φ}) ⊢ ⊥))
```

```
def Set.witnessed (Γ : Set Form) : Prop := ∀  {φ : Form},
  φ ∈ Γ →
    match φ with
      | ex x, ψ => ∃  i : NOM, ψ[i // x] ∈ Γ
      | _   => φ ∈ Γ
```

5.2.2 Main Proofs

In this section, we provide a brief overview on the way we used Lean to prove the main properties shown in the mathematical section of this thesis.

Soundness

The file Soundness.lean contains the main proofs necessary for Soundness. Lemmas 3.0.4, 3.0.5, 3.0.6 proved earlier correspond to theorems `generalize_not_free`, `svar_substitution` and `nom_substitution`, respectively. For brevity of exposition, we only mention the statements of these theorems:

```
theorem generalize_not_free (h1 : is_free v φ = false) : ⊨ (φ ↔
    (all v, φ))

theorem svar_substitution {φ : Form} {x y : SVAR} {M : Model} {s
    : M.W} {g g' : I M.W}
  (h_subst : is_substable φ y x) (h_var : is_variant g g' x)
    (h_which_var : g' x = g y) :

  (((M,s,g) ⊨ φ[y // x]) ↔ (M,s,g') ⊨ φ)

theorem nom_substitution {φ : Form} {x : SVAR} {i : NOM} {M :
    Model} {s : M.W} {g g' : I M.W}
    (h_var : is_variant g g' x) (h_which_var : g' x = M.Vₙ i) :

    (((M,s,g) ⊨ φ[i // x]) ↔ ((M,s,g') ⊨ φ))
```

Proposition 2.2.1 is formalized as:

```
theorem sat_iterated_nec {φ : Form} {n : Nat} {M : Model} {s :
    M.W} {g : I M.W} :
  ((M,s,g) ⊨ iterate_nec n φ) ↔ (∀  s' : M.W, (path M.R s s' n)
    → (M,s',g) ⊨ φ)

theorem sat_iterated_pos {φ : Form} {n : Nat} {M : Model} {s :
    M.W} {g : I M.W} :
  ((M,s,g) ⊨ iterate_pos n φ) ↔ (∃  s' : M.W, (path M.R s s' n)
    ∧ (M,s',g) ⊨ φ)
```

Our treatment of tautologies (Lemma 3.0.2) corresponds to the following lines of Lean code. The function e_{Sat} we defined in natural language is the function `model_val_func` below:

```
noncomputable def model_val_func (M : Model) (s : M.W) (g : I M.W)
    : Form → Bool := λ φ => ite ((M,s,g) ⊨ φ) true false

noncomputable def model_eval (M : Model) (s : M.W) (g : I M.W) :
    Eval :=
     let f := model_val_func M s g
     have p1 : f ⊥ = false := by simp [model_val_func]
     have p2 : ∀  φ ψ : Form, (f (φ → ψ) = true) ↔ (¬(f φ) =
    true ∨ (f ψ) = true) := λ φ ψ : Form => by simp
    [model_val_func]
     ⟨f, p1, p2⟩

theorem taut_sound : Tautology φ → ⊨ φ := by
  intro h M s g
  have := h (model_eval M s g)
  simp [model_eval, model_val_func] at this
  exact this
```

Notice the `noncomputable` keyword in the definition of `model_val_func` and `model_eval`. We apply an if-then-else statement to `(M,s,g) ⊨ φ`, but we have not provided a proof that this proposition is decidable. The Lean compiler is then forced to produce non-executable code for these definitions.

By induction on formulas, we are then able to prove Theorem 3.0.7. The details of our proof in Lean mirror exactly the steps we followed in natural language, so we will only give the statement of the theorem we proved:

```
theorem WeakSoundness : (⊢ φ) → (⊨ φ)
```

Full soundness (Theorem 3.0.8) then follows easily as well:

```
theorem Soundness : (Γ ⊢ φ) → (Γ ⊨ φ) := by
  rw [SyntacticConsequence]
  intro h
  apply SetEntailment
  match h with
  | ⟨L, conseq⟩ =>
    have := (@WeakSoundness (conjunction Γ L→φ)) conseq
    exact ⟨L, this⟩
```

Deduction

We turn to the file ProofUtils.lean, where we prove the theorems and metatheorems needed for completeness. We state Deduction (Theorem 2.3.2) in the usual way:

```
theorem Deduction (Γ : Set Form) : Γ ⊢ (ψ → φ) ↔ (Γ ∪ {ψ}) ⊢ φ
```

The proof is also standard. We need however some help to show that any conjunction of formulas from $\Gamma \cup \{\psi\}$ is a conjunction of formulas from Γ; and that any conjunction of formulas from $\Gamma \cup \{\psi\}$ that doesn't include ψ is a conjunction of formulas from Γ. These should be trivial facts, but our definition of conjunction has the disadvantage lists of Γ-formulas are a different type than lists of $\Gamma \cup \{\psi\}$-formulas. The required coercions and the proofs of the two facts are done in the file ListUtils.lean. The proof of Deduction is, then:

```
theorem Deduction (Γ : Set Form) : Γ ⊢ (ψ → φ) ↔ (Γ ∪ {ψ}) ⊢ φ
     := by
  apply Iff.intro
  . intro h
    match h with
    | ⟨L, hpf⟩ =>
        have t_com12 := tautology (@com12 (conjunction Γ L) ψ φ)
        have l1 := mp t_com12 hpf
        have t_imp := tautology (@imp ψ (conjunction Γ L) φ)
        have l2 := mp t_imp l1
        have pfmem : ψ ∈ Γ ∪ {ψ} := by simp
```

```
      let L' : List ↑(Γ ∪ {ψ}) := ⟨ψ, pfmem⟩ :: list_convert L
      rw [conj_incl] at l2
      exact ⟨L', l2⟩
  . intro h
    match h with
    | ⟨L', hpf⟩ =>
      have t_ax1 := tautology (@ax_1 (conjunction (Γ ∪ {ψ})
    L'→φ) ψ)
      have l1 := mp t_ax1 hpf
      have t_com12 := tautology (@com12 ψ (conjunction (Γ ∪ {ψ})
    L') φ)
      have l2 := mp t_com12 l1
      by_cases elem : elem' L' ψ
      . have t_help := tautology (deduction_helper L' ψ (ψ→φ)
    elem)
        have l3 := mp t_help l2
        have t_idem := tautology (@idem (conjunction (Γ ∪ {ψ})
    (filter' L' ψ)) ψ φ)
        have l4 := mp t_idem l3
        have not_elem_L' := eq_false_of_ne_true (@filter'_filters Γ
     ψ L')
        let L : List Γ := list_convert_rev (filter' L' ψ)
    not_elem_L'
        rw [conj_incl_rev (filter' L' ψ) not_elem_L'] at l4
        exact ⟨L, l4⟩
      . have elem : elem' L' ψ = false := by simp only [elem]
        let L : List Γ := list_convert_rev L' elem
        rw [conj_incl_rev L' elem] at l2
        exact ⟨L, l2⟩
```

Hybrid Derivations

In the rest of the file ProofUtils.lean, we prove all the facts given in section 2.3.1, and also all properties of MCS's as stated at Proposition 4.1.1. Many of these theorems depend on properties of substitutions that we proved in the file Substitutions.lean.

To illustrate the relationship between the formal derivations given earlier and their Lean-counterpart, consider the proof we have given for Proposition 2.3.7.(i). We employed a somewhat even-handed mixture of tactic-based proving (in order to more efficiently argue by contraposition) and term-based proving.

```
lemma b361 : ⊢ ((φ → ex x, ψ) → ex x, (φ → ψ)) := by
  apply mp
  . apply tautology
    apply contrapositive'
  . simp only [←Γ_empty, Deduction, Set.union_singleton,
    insert_emptyc_eq]
    let Γ : Set Form := {∼(ex x, φ→ψ)}
    have l1 : Γ ⊢ (∼(ex x, φ→ψ)) := by apply Γ_premise; simp
    rw [Form.bind_dual] at l1
    have l2 := Γ_theorem (tautology (@dne (all x, ∼(φ→ψ)))) Γ
    have l3 := Γ_mp l2 l1
    have l4 := Γ_theorem (@ax_q2_svar_instance x (∼(φ→ψ))) Γ
    have l5 := Γ_mp l4 l3
    have l6 := Γ_theorem (tautology (taut_iff_mp (@imp_neg φ ψ)))
    Γ
    have l7 := Γ_mp l6 l5
    have l8 := Γ_conj_elim_l l7
    have l9 := Γ_conj_elim_r l7
    have l10 : Γ ⊢ (∼(ex x, ψ)) := by
      rw [Form.bind_dual]
      apply Γ_mp; apply Γ_theorem; apply tautology; apply dni
      apply Γ_univ_intro'
      . simp [is_free, -implication_disjunction]
      . exact l9
    have l11 := Γ_conj_intro l8 l10
    have l12 := Γ_mp (Γ_theorem (tautology (taut_iff_mpr
    (@imp_neg φ (ex x, ψ)))) Γ) l11
    exact l12
```

Worth mentioning is that we proved a version of Lemma 4.1.6 which allows us to rename any nominal in a formula by some nominal that does not occur in it. We did not need this property in the mathematical section, but it should become clear shortly why we need it in Lean. Theorem `generalize_constants` referred to in the following proof is our formalization of Lemma 4.1.6.

```
theorem rename_constants (j i : NOM) (h : nom_occurs j φ = false)
    : ⊢ φ ↔ ⊢ (φ[j // i]) := by
  apply Iff.intro
  . intro pf
    let x := φ.new_var
    have x_geq : x ≥ φ.new_var := by simp; apply Nat.le_refl
    have l1 := generalize_constants i x_geq pf
    have l2 := ax_q2_nom (φ[x // i]) x j
    have l3 := mp l2 l1
    have : φ[x//i][j//x] = φ[j//i] := svar_svar_nom_subst x_geq
    rw [this] at l3
    exact l3
  . intro pf
    let x := (φ[j//i]).new_var
    have x_geq : x ≥ (φ[j//i]).new_var := by simp; apply
    Nat.le_refl
    have l1 := generalize_constants j x_geq pf
    have : φ[j//i][x//j] = φ[x//i] := dbl_subst_nom i h
    rw [this] at l1
    have l2 := ax_q2_nom (φ[x // i]) x i
    have l3 := mp l2 l1
    rw [←eq_new_var] at x_geq
    have : φ[x//i][i//x] = φ[i//i] := svar_svar_nom_subst x_geq
    rw [nom_subst_self] at this
    rw [this] at l3
    exact l3
```

Model Existence

In the file Completeness.lean, we prove Theorem 4.0.1. We first define the two statements of completeness:

```
def completeness_statement := (∀  (Γ : Set Form) (φ : Form), Γ ⊨
    φ → Γ ⊢ φ)
def cons_sat_statement     := (∀  (Γ : Set Form), Γ.consistent →
    Γ.satisfiable)
```

Model existence is, naturally, the statement of their equivalence:

```
theorem ModelExistence : completeness_statement ↔
    cons_sat_statement
```

Propositions 2.2.2, 2.2.3 and 2.3.8 correspond to theorems `satisfiable_iff_nocontradiction`, `unsatisfiable_iff_contradiction` and `notprove_consistentnot`:

```
theorem satisfiable_iff_nocontradiction (Γ : Set Form) : Γ
    .satisfiable ↔ Γ ⊭ ⊥

theorem unsatisfiable_iff_contradiction (Γ : Set Form) : ¬Γ
    .satisfiable ↔ Γ ⊨ ⊥

theorem notprove_consistentnot : (Γ ⊬ φ) ↔ (Γ ∪ {∼φ}).consistent
```

Countability

Our proof of Proposition 4.1.2, among others, relied on the countability of hybrid language. While no explicit proof has been given in the mathematical part of this thesis, we explained the proof is standard and we referred the reader to [21]. Lean, however, is not so easily convinced by bibliographical references. In order to prove all the propositions we need, we are also forced to explicitly formalize the proof that $\mathscr{L}(\forall)$ is countable. We have accomplished this in the file FormCountable.lean. We will proceed to give a brief explanation of the contents of this file.

First of all, a clarification of terms is necessary. In Mathlib ([24]), the Countable α type class is defined as a wrapper around the Prop that there exists an injective function from α to $\mathbb{N}$:

```
class Countable (α : Sort u) : Prop where
  /-- A type ‘α‘ is countable if there exists an injective map ‘α
    → ℕ‘. -/
  exists_injective_nat' : ∃  f : α → ℕ, Injective f
```

This differs from both Encodable and Denumerable. Encodable ([30]), on the one hand, requires the injection to be explicitly constructed (not by a proof of existence), along with its partial inverse from $\mathbb{N}$ to $Option\,\alpha$.

```
class Encodable (α : Type _) where
  /-- Encoding from Type α to ℕ -/
  encode : α → ℕ
  /-- Decoding from ℕ to Option α-/
  decode : ℕ → Option α
  /-- Invariant relationship between encoding and decoding-/
  encodek : ∀  a, decode (encode a) = some a
```

Hence Encodable is the constructive, and thus *stronger*, counterpart of Countable. Mathlib does have the definition necessary to produce an Encodable from a Countable, but this function is noncomputable, since Nonempty.some and ofInj rely on Choice under the hood to synthesize the explicit functions:

```
noncomputable def ofCountable (α : Type _) [Countable α] :
    Encodable α :=
  Nonempty.some <|
    let ⟨f, hf⟩ := exists_injective_nat α
    ⟨ofInj f hf⟩
```

Denumerable ([29]), on the other hand, is an even stronger version of Encodable, requiring the mapping from α to $\mathbb{N}$ to be a bijection.

```
class Denumerable (α : Type _) extends Encodable α where
  /-- `decode` and `encode` are inverses. -/
  decode_inv : ∀  n, ∃  a ∈ decode n, encode a = n
```

Out of these three possibilities, we opted to implement an instance of Countable Form. To understand why, we first note that our proofs do not require a full-fledged bijection between formulas and natural numbers. Our only requirement was being able to list *all* formulas by an enumeration, i.e. by a function from $\mathbb{N}$ to Form. There never occurs the necessity to enumerate each formula *only once*. This implies that it is sufficient to rely only on the *surjectivity* of the enumeration. Now, classically, the existence of a surjective function from $\mathbb{N}$ to Form is guaranteed by the existence of an injective function from Form to $\mathbb{N}$. Constructively, this is not the case: both functions must be explicitly defined, along with the proofs of their respective properties. For the purposes of this formalization, however, classical reasoning suffice. We decided to only show that there is an injection between Form and $\mathbb{N}$, and to noncomputably synthesize a surjection in the opposite direction (i.e., an enumeration) when needed.

Having restricted the scope of our proof, we now consider the exact manner in which we intend to show that there exists an injection. It seems clear that we must define one, but how do we go about it? Two choices seem plausible at first. We could attempt to write a gödelization function, using prime factorizations; or we could define a function to represent Form as String, and rely on String's countability. We found disadvantages with both approaches. First off, the reliance of Gödel functions on the Fundamental Theorem of Arithmetic would have been inconvenient, as we found the Mathlib statement of the Theorem ([27]) to be somewhat idiosyncratic and ill-adapted to our intended use. Second, there is no instance of Countable String defined in Mathlib (since Strings are Lists of Chars, and the type Char is primarily intended for programming, not proving), and writing a function to parse Strings intro Forms would have been a rather tedious task. So we chose a third option: we defined an injection of Form into List $\mathbb{N}^4$, and we relied on the countability of List $\mathbb{N}^4$. We give a brief explanation of our reasoning below.

```
def pow2list (l : List (ℕ × ℕ × ℕ × ℕ)) := List.map (λ
    (a,b,c,d) => (2^(a+1), 2^(b+1), 2^(c+1), 2^(d+1))) l

def pow3list (l : List (ℕ × ℕ × ℕ × ℕ)) := List.map (λ
    (a,b,c,d) => (3^(a+1), 3^(b+1), 3^(c+1), 3^(d+1))) l

def squash (n m : List (ℕ × ℕ × ℕ × ℕ)) : List (ℕ × ℕ × ℕ × ℕ
    ) := pow2list n ++ pow3list m

def Form.encode : Form → List (ℕ × ℕ × ℕ × ℕ)
  | Form.bttm    => [(0,0,0,1)]
  | Form.prop p  => [(0,0,p.letter+1,0)]
  | Form.svar x  => [(0,x.letter+1,0,0)]
  | Form.nom i   => [(i.letter+1,0,0,0)]
  | Form.impl φ ψ   => [(0,0,0,2)] ++ (squash φ.encode ψ.encode)
  | Form.box φ    => [(0,0,0,3)] ++ φ.encode
  | Form.bind x φ=> [(0,0,0,4), (0,x.letter+1,0,0)] ++ φ.encode
```

The main definition here is that of Form.encode. Each *atomic formula* is mapped to a singleton list of four-dimensional natural numbers. We use three dimensions corresponding to the three kinds terms (PROP, SVAR or NOM), and we use the fourth dimension to signify the kind of connective by which the formula is introduced (by this we mean $\longrightarrow$, $\Box$ and *all* x; we also include $\perp$ in this dimension). Whole formulas then correspond to lists of $\mathbb{N}^4$. Take note of the additional function *squash* we used for the definition of implication. Its behaviour is to take two lists, raise the first one to the power of 2, then append it to the second one raised to the power of 3. Since implication is a binary connective, we use this function to ensure that the left-hand-side of the implication is clearly delimited from the right-hand-side. Think of this as a mathematical way to parenthesize formulas. The injectivity of Form.encode hinges upon the injectivity of *squash*, which in turn hinges upon the fact that no positive power of 2 is equal to any positive power of 3. The heart of the proof is this lemma, which states that given any two $\mathbb{N}^4$ lists a and b, if $squash(a,b) = squash(n,m)$, then $pow2list(a) = pow2list(n)$ and $pow3list(b) = pow3list(m)$ (assuming without loss of generality that length of a is smaller than the length of b).

```
lemma squash_lemma_wlog (h : (pow2list a).length ≤ (pow2list
   n).length) : squash a b = squash n m → (pow2list a = pow2list
   n ∧ pow3list b = pow3list m) := by
  intro hyp
  simp [squash] at hyp
  have by_l1 := sum_is_prefix hyp h
  have by_l2 := split_prefix_suffix by_l1
  match by_l2 with
  | ⟨suf, hsuf⟩ =>
      clear h by_l1 by_l2
      simp [hsuf] at hyp
      cases suf
      . simp at hyp hsuf
        exact ⟨Eq.symm hsuf.left, hyp⟩
      . exfalso
        have is_pow_2 := suffix_pow2 hsuf.left
        cases b
        . simp [pow3list] at hyp
        . simp [pow3list, Prod.eq_iff_fst_eq_snd_eq] at hyp
          have abs_1 := hyp.left.left
          match is_pow_2 with
          | ⟨n, abs_2⟩ =>
              rw [abs_2] at abs_1
              apply prime_2_3
              apply abs_1
```

Defining an instance of Countable Form then becomes an easy matter:

```
theorem Inject_Form : Form.encode.Injective := by
  intro φ ψ
  intro h
  induction φ generalizing ψ with
  | impl a b ih1 ih2  =>
      cases ψ <;> simp [Form.encode, -implication_disjunction] at
    *
      apply And.intro <;> (first | apply ih1 | apply ih2) <;> simp
    [squash_inj h]
  | box φ ih  =>
      cases ψ <;> first | simp [Form.encode,
    -implication_disjunction] at *; try apply ih; try assumption
  | bind x φ ih  =>
      cases ψ <;> simp [Form.encode, -implication_disjunction] at
    *
      apply And.intro
      . exact ih h.right
      . exact congrArg SVAR.mk h.left
  | _  =>
      induction ψ <;> simp [Form.encode] at * <;>
        first | exact congrArg PROP.mk h | exact congrArg SVAR.mk
    h |
        try exact congrArg NOM.mk h

instance : Countable Form := Inject_Form.countable
```

Language Expansions

We used language expansions to prove the extended version of Lindenbaum's Lemma (4.1.9). This presents us with yet another challenge. Working in set theory as a metalanguage, there is never a scarcity of objects: we can always find infinitely many new nominal symbols to expand a language's signature with. Say we are working in a signature $\langle \text{PROP}, \text{SVAR}, \text{NOM} \rangle$, where we take $\text{NOM} := \mathbb{N}$. Set-theoretically, expanding NOM to $\mathbb{Z}$ would be a very suitable choice in order to apply Lemma 4.1.9. Things are however not as simple in Lean's type theory. Any type α already encompasses all of its possible terms, and thus all of its possible expansions. Even if we have a term a of type `Set` α, it is not true that there always exists some a' such that $a \subset a'$. For take the total set, `λ _ => True`. As per the Mathlib documentation, "Sets in Lean are homogeneous; all their elements have the same type" [28]. All terms of type α are members of its total set. There is no way to expand it.

We identified two possible ways out of this difficulty. The first one is to give a parallel definition for the type Form, in which nominals are defined as terms of a more encompassing type. For example:

```
inductive Form' where
  | bttm : Form'
  | prop : PROP  → Form'
  | svar : SVAR  → Form'
  | nom  :    ℤ  → Form'
  | impl : Form' → Form' → Form'
  | box  : Form' → Form'
  | bind : SVAR  → Form' → Form'
```

Notice the line `| nom : ℤ → Form'`; the definition of Form provided earlier has `| nom : ℕ → Form`. The next steps would be to provide a coercion from Form to Form', and show that all the theorems and validities of Form are also theorems and validities of Form'. Then we would be able to prove a form of the Lindenbaum Lemma stating that, given any consistent set of Form, its coercion to a set of Form' is included in a witnessed maximal consistent set of Form'.

We feel this solution would be undesirable. We intend that the statements we prove are as general as possible. However, by defining a parallel hybrid language Form', none of the properties we proved for Form would hold by default for the new language Form'. In particular, if we wanted to prove Lindenbaum's Lemma for Form', we would have to define yet another type Form"; and then prove a completely new statement of the lemma, which would now link Form' and Form". We believe that this does not reflect the content of the set-theoretic statement we proved under Lemma 4.1.9. For Lemma 4.1.9 can be applied to any language whatsoever, after any number of expansions as defined by Definition 4.1.3.

We have thus decided to turn to the *internal* properties of the language we defined. At this point, it becomes crucial that we defined the set of nominals to be denumerably infinite, more specifically in a bijection with $\mathbb{N}$. In what follows, we outline the technique we developed, and the difficulties we still faced. Encouragingly, our new difficulties are tractable: they are only a matter of filling in some missing proofs. They do not involve the limits of Lean's type theory, as our original conundrum did.

Let φ be a formula and φ^{odd} be the formula obtained by substituting all nominals i which occur in φ by $2*i+1$. NOM's denumerability ensures us that such a $2*i+1$ always exists. Then, (1) it is true that $\vdash \varphi$ iff $\vdash \varphi^{odd}$. Furthermore, letting Γ be a set of formulas, we define $\Gamma^{odd} := \{\varphi^{odd} \mid \varphi \in \Gamma\}$. Then, (2) it is true that $\Gamma \vdash \psi$ iff $\Gamma^{odd} \vdash \psi^{odd}$.

Provided we can prove them, these are two highly significant statements. There always exists a denumerable infinity of nominals which do not occur in Γ^{odd}, regardless of what the original set Γ is. Namely, all the *even* nominals, of the form $2*i$. Uniformly mapping all the nominals into odd nominals therefore amounts to an expansion of the language: we *make room* for an infinity of fresh nominals. Statements (1) and (2) ensure us that such mappings are theorem-preserving. They are the reflection of Proposition 4.1.7 that we have proved set-theoretically.

Some of the work in the file Substitutions.lean is directed towards proving properties required for these equivalences. In the file ProofUtils.lean, we have managed to provide a complete formalization only of (1). The current version of our formalization still relies on a `sorry` statement for the proof of (2).

We defined φ^{odd} and Γ^{odd} in such a way as to make our proof of equivalence (1) easier. Our aim was to define φ^{odd} as a recursive substitution, such that at each iteration, the odd nominal to be inserted does not occur in the formula obtained up to that point. Thus, by the theorem `rename_constants` we mentioned earlier, each iteration of this recursive procedure would be theorem-preserving.

For this purpose, we first generalize the notion of substitution. We define a way to substitute *lists* of nominals for other *lists* of nominals, and call it *bulk* *substitution*. This function takes a formula and two lists as arguments (first the new list, then the old list of nominals), and outputs another formula:

```
def Form.bulk_subst : Form → List NOM → List NOM → Form
| φ, h₁ :: t₁, h₂ :: t₂ => bulk_subst (φ[h₁ // h₂]) t₁ t₂
| φ, _, []     =>  φ
| φ, [], _     =>  φ
```

Next, we need to do two things: one, produce a list of all nominals in a formula φ; two, map this list by the function `(λ i => 2*i+1)`. We insist on making sure the list of nominals is strictly decreasing. On the one hand, this is our guard against accidental substitution; and on the other hand, its decreasing property allows for the recursive application of theorem `rename_constants`, as we set up to do. For this purpose, we used Mathlib's mergesort algorithm [26], along with the function List.dedup [25] to ensure uniquness of elements in the list.

```
def Form.list_noms : Form → List NOM
| nom  i   => [i]
| impl φ ψ => (List.merge GE.ge φ.list_noms ψ.list_noms).dedup
| box φ     => φ.list_noms
| bind _ φ => φ.list_noms
| _         => []


def Form.odd_list_noms : Form → List NOM := λ φ => φ
    .list_noms.map (λ i => 2*i+1)


def Form.odd_noms : Form → Form := λ φ => φ.bulk_subst φ
    .odd_list_noms φ.list_noms


def Set.odd_noms : Set Form → Set Form := λ Γ => {Form.odd_noms φ
     | φ ∈ Γ}
```

With these definitions on hand, and after showing that we can indeed recursively apply `rename_constants`, we are able to prove in ProofUtils.lean:

```
theorem pf_odd_noms : ⊢ φ ↔ ⊢ φ.odd_noms


theorem pf_odd_noms_set : Γ ⊢ φ ↔ Γ.odd_noms ⊢ φ.odd_noms
```

At the moment, `pf_odd_noms_set` still depends on this statement left unproved:

```
theorem odd_impl : (φ ⟶ ψ).odd_noms = φ.odd_noms ⟶ ψ.odd_noms
```

Lindenbaum's Lemma

For clarity, in section 4.1 we gave two different statements and two different proofs of Lindenbaum's Lemma: Lemma 4.1.3 and Lemma 4.1.3. For brevity, in Lean we define only *one* construction of a sequence of sets. Based on that single construction, we prove both statements of the lemma.

First, we define how to obtain Γ_{i+1} from Γ_i, given a formula φ:

```
def lindenbaum_next (Γ : Set Form) (φ : Form) : Set Form :=
  if (Γ ∪ {φ}).consistent then
    match φ with
    | ex x, ψ =>
        if c : ∃  i : NOM, all_nocc i (Γ ∪ {φ}) then
          Γ ∪ {φ} ∪ {ψ[c.choose // x]}
        else
          Γ ∪ {φ}
    | _        =>  Γ ∪ {φ}
  else
    Γ
```

`all_nocc` is a Prop stating that `i` does not occur in any of the formulas from a given set:

```
def all_nocc (i : NOM) (Γ : Set Form) := ∀  (φ : Form), φ ∈ Γ ⟶
    nom_occurs i φ = false
```

Thus, our definition of `lindenbaum_next` already adds a witness to all existential formulas, if there exists an available nominal in the language. For regular Lindenbaum, this design decision has no significance. For extended Lindenbaum, we will show that such a nominal *always* exists if we start with $\Gamma_0 := \Gamma^{odd}$.

Now we define the whole indexed family $(\Gamma_i)_{i\in\mathbb{N}}$. Usually, the enumeration of formulas starts from 1 ($\varphi_1, \varphi_2, \ldots$), and we take $\Gamma_0 := \Gamma$. However, in Lean it's much tidier to enumerate from 0 ($\varphi_0, \varphi_1, \ldots$), and take $\Gamma_0 := \Gamma \cup \{\varphi_0\}$ if it is consistent, and $\Gamma_0 := \Gamma$ otherwise.

```
def lindenbaum_family (enum : Nat → Form) (Γ : Set Form) : Nat →
    Set Form
| .zero   => lindenbaum_next Γ (enum 0)
| .succ n =>
     let prev_set := lindenbaum_family enum Γ n
     lindenbaum_next prev_set (enum (n+1))

notation Γ "(" i "," e ")" => lindenbaum_family e Γ i
```

Note that `lindenbaum_family` is dependent upon an enumeration `enum : Nat → Form`.

Next, we define the supremum of this sequence of sets:

```
def LindenbaumMCS (enum : Nat → Form) (Γ : Set Form) (_ : Γ
    .consistent) : Set Form :=
     {φ | ∃  i : Nat, φ ∈ Γ (i, enum)}
```

Our goal now is to show that `LindenbaumMCS` is a (witnessed) MCS. (Note the anonymous argument `(_ : Γ.consistent)`. This is a sanity check to ensure `LindenbaumMCS` is not applied to anything other than consistent sets.)

We then prove various lemmas related to Lindenbaum. Among these, we prove that given an injective mapping `f : Form → ℕ`, its left-inverse `e` and some consistent set Γ; for any a finite list L of elements in `LindenbaumMCS e Γ c`, there is a natural i such that all elements in L occur in Γ_i. More specifically, $i = \max_{\varphi \in L} \mathtt{f}(\varphi)$. This is simply the formal way of saying that any finite subset of `LindenbaumMCS` must be a subset of Γ_i, with i being the number corresponding to the "greatest" formula of that subset. We used this fact to show consistency in the proof of Lemma 4.1.3 (though without specifically characterizing i), and its purpose is the same in Lean. The formal statement we have proved is:

```
lemma list_at_finite_step {Γ : Set Form} {c : Γ.consistent} (f :
    Form → ℕ) (f_inj : f.Injective) (e : ℕ → Form) (e_inv : e =
    f.invFun) (L : List (LindenbaumMCS e Γ c)) :
     {↑φ | φ ∈ L} ⊆ (Γ (L.max_form f, e))
```

Where `L.max_form` refers to a function which calculates the "greatest" element in `L`, given a mapping `Form → ℕ`.

After proving all required lemmas, we prove Lindenbaum:

```
theorem RegularLindenbaumLemma : ∀  Γ : Set Form, Γ.consistent →
   ∃  Γ' : Set Form, Γ ⊆ Γ' ∧ Γ'.MCS := by
  intro Γ cons
  let ⟨f, f_inj⟩ := exists_injective_nat Form
  let enum       := f.invFun
  let Γ' := LindenbaumMCS enum Γ cons
  have enum_inv : enum = f.invFun := rfl
  exists Γ'
  apply And.intro
  . -- Γ is included in Γ'
    let Γ₀ := Γ (0, enum)
    have Γ_in_Γ₀ : Γ ⊆ Γ₀ := Γ_in_family
    have Γ₀_in_family := @all_sets_in_family enum Γ cons 0
    rw [show LindenbaumMCS enum Γ cons = Γ' by simp, show Γ (0,
   enum) = Γ₀ by simp] at Γ₀_in_family
    intro _ φ_in_Γ
    exact Γ₀_in_family (Γ_in_Γ₀ φ_in_Γ)
  . rw [Set.MCS]
    apply And.intro
    . exact LindenbaumConsistent cons f_inj enum_inv
    . intro φ
      exact LindenbaumMaximal cons f_inj enum_inv φ
```

Two facts are essential here: first, that we have access to an injective mapping `f : Form → ℕ`. We took care of this earlier by writing an instance of Countable Form; the function `exists_injective_nat` then provides us with the required function. And second, that we have access to a function `enum : ℕ → Form`, such that `enum (f φ) = φ`; i.e., `enum` is the left-inverse of `f`. Mathlib also takes care of this, by the function `Function.invFun`.

The proof for the expanded version of Lindenbaum's Lemma still needs to be filled. However, it shares most of the intermediary lemmas it needs with the regular lemma, which we have already proved.

Truth Lemma

We give a brief overview on our completed model construction. It is to be found in the file CompletedModel.lean.

We first define a notion of general models, in which the valuations of state symbols are no longer constrained to be singleton sets:

```
structure GeneralModel where
  W : Type
  R : W → W → Prop
  Vₚ: PROP → Set W
  Vₙ: NOM  → Set W

def GeneralI (W : Type) := SVAR → Set W
```

We define the canonical model as a general model. To simplify the process, our decision was to define the type `W` to be `Set Form`; i.e., the set of states in the canonical model is the set of all sets of formulas, not the set of all MCS's. We then constrain the accesibility relation `R` to hold only between MCS's. Only when we get to the definition of the (standard) completed model will we explicitly restrict `W`.

```
def Canonical : GeneralModel where
  W := Set Form
  R := restrict_by Set.MCS (λ Γ => λ Δ => (∀  {φ : Form}, □φ ∈
    Γ → φ ∈ Δ))
  Vₚ:= λ p => {Γ | Γ.MCS ∧ ↑p ∈ Γ}
  Vₙ:= λ i => {Γ | Γ.MCS ∧ ↑i ∈ Γ}

def CanonicalI : SVAR → Set (Set Form) := λ x => {Γ | Γ.MCS ∧ ↑x
    ∈ Γ}
```

We then define generated submodels of the canonical model. We use the additional parameter `restriction : Set Form → Prop` to restrict the sets in the generated submodel by some given predicate. In particular, we define witnessed models as generated submodels restricted by `Set.witnessed`.

```
def Set.GeneratedSubmodel (Θ : Set Form) (restriction : Set Form
    → Prop) : GeneralModel where
  W := Set Form
  R := λ Γ => λ Δ =>
    (∃  n, path (restrict_by restriction Canonical.R) Θ Γ n) ∧
    (∃  m, path (restrict_by restriction Canonical.R) Θ Δ m) ∧
    Canonical.R Γ Δ
  Vₚ:= λ p => {Γ | (∃  n, path (restrict_by restriction
    Canonical.R) Θ Γ n) ∧ Γ ∈ Canonical.Vₚ p}
  Vₙ:= λ i => {Γ | (∃  n, path (restrict_by restriction
    Canonical.R) Θ Γ n) ∧ Γ ∈ Canonical.Vₙ i}

def Set.GeneratedSubI (Θ : Set Form) (restriction : Set Form →
    Prop) : GeneralI (Set Form) := λ x =>
  {Γ | (∃  n, path (restrict_by restriction Canonical.R) Θ Γ n) ∧
    Γ ∈ CanonicalI x}
```

Thus, witnessed models become:

```
def WitnessedModel {Θ : Set Form} (_ : Θ.MCS) (_ : Θ.witnessed)
    : GeneralModel := Θ.GeneratedSubmodel Set.witnessed
def WitnessedI {Θ : Set Form} (_ : Θ.MCS) (_ : Θ.witnessed) :
    GeneralI (Set Form) := Θ.GeneratedSubI Set.witnessed
```

For the definition of completed models, we use $\{\bot\}$ as our dummy state.

```
def CompletedModel {Θ : Set Form} (mcs : Θ.MCS) (wit : Θ
    .witnessed) : GeneralModel where
  W := Set Form
  R := λ Γ => λ Δ => ((WitnessedModel mcs wit).R Γ Δ) ∨ (Γ =
    {Form.bttm} ∧ Δ = Θ)
  Vₚ:= λ p => (WitnessedModel mcs wit).Vₚ p
  Vₙ:= λ i => if (WitnessedModel mcs wit).Vₙ i ≠ ∅
              then  (WitnessedModel mcs wit).Vₙ i
              else { {Form.bttm} }
def CompletedI {Θ : Set Form} (mcs : Θ.MCS) (wit : Θ.witnessed)
    : GeneralI (Set Form) := λ x =>
  if (WitnessedI mcs wit) x ≠ ∅
              then  (WitnessedI mcs wit) x
              else { {Form.bttm} }
```

We prove the Lean counterpart of Lemmas 4.2.3 and 4.2.4:

```
lemma subsingleton_valuation : ∀  {Θ : Set Form} {R : Set Form →
    Prop} (i : NOM), Θ.MCS → ((Θ.GeneratedSubmodel R).Vₙ
    i).Subsingleton
```

```
lemma subsingleton_i : ∀  {Θ : Set Form} {R : Set Form → Prop}
    (x : SVAR), Θ.MCS → ((Θ.GeneratedSubI R) x).Subsingleton
```

This allows us to show that the completed model indeed maps state symbols to singleton sets:

```
lemma completed_singleton_valuation {Θ : Set Form} (mcs : Θ.MCS)
    (wit : Θ.witnessed) (i : NOM) : ∃  Γ : Set Form,
    (CompletedModel mcs wit).Vₙ i = {Γ} := by
  simp [CompletedModel]
  split
  . simp
  . next h =>
      rw [←ne_eq, ←Set.nonempty_iff_ne_empty, Set.nonempty_def]
    at h
      match h with
      | ⟨Γ, h⟩ =>
          exists Γ
          apply (Set.subsingleton_iff_singleton h).mp
          apply wit_subsingleton_valuation
          assumption

lemma completed_singleton_i {Θ : Set Form} (mcs : Θ.MCS) (wit : Θ
    .witnessed) (x : SVAR) : ∃  Γ : Set Form, (CompletedI mcs
    wit) x = {Γ} := by
  simp [CompletedI]
  split
  . simp
  . next h =>
      rw [←ne_eq, ←Set.nonempty_iff_ne_empty, Set.nonempty_def]
    at h
      match h with
      | ⟨Γ, h⟩ =>
          exists Γ
          apply (Set.subsingleton_iff_singleton h).mp
          apply wit_subsingleton_i
          assumption
```

And so we are finally able to define completed models as standard models.

To prove the existential clause of the truth lemma, it becomes crucial that we finally constrain `W` to only include MCS's or the dummy state {⊥}. We use subtyping to impose this restriction. Now, we must also ensure that the dummy state is a member of `W` *only if it is needed*. We employ the follwoing trick:

```
def Set.MCS_in (Γ : Set Form) {Θ : Set Form} (mcs : Θ.MCS) (wit :
    Θ.witnessed) : Prop := ∃  n, path (WitnessedModel mcs wit).R
    Θ Γ n

def needs_dummy {Θ : Set Form} (mcs : Θ.MCS) (wit : Θ.witnessed)
    := (∃  i, ((CompletedModel mcs wit).Vₙ i) = { (Set.singleton
    Form.bttm) }) ∨ (∃  x, ((CompletedI mcs wit) x) = {
    (Set.singleton Form.bttm) })

def Set.is_dummy (Γ : Set Form) {Θ : Set Form} (mcs : Θ.MCS) (wit
    : Θ.witnessed) := has_dummy mcs wit ∧ Γ = {Form.bttm}

noncomputable def StandardCompletedModel {Θ : Set Form} (mcs : Θ
    .MCS) (wit : Θ.witnessed) : Model :=
    ⟨{Γ : Set Form // Γ.MCS_in mcs wit ∨ Γ.is_dummy mcs wit},
      λ Γ => λ Δ => (CompletedModel mcs wit).R Γ.1 Δ.1,
      λ p => {Γ | Γ.1 ∈ ((CompletedModel mcs wit).Vₚ p)},
      λ i => ⟨(completed_singleton_valuation mcs wit i).choose,
    choose_subtype mcs wit⟩⟩

noncomputable def StandardCompletedI {Θ : Set Form} (mcs : Θ.MCS)
    (wit : Θ.witnessed) : I (StandardCompletedModel mcs wit).W :=
    λ x => ⟨(completed_singleton_i mcs wit x).choose,
    choose_subtype' mcs wit⟩
```

More explicitly, we define `W` to be the set of sets of formulas which either satisfy `Set.MCS_in`, or `Set.is_dummy`. `Set.MCS_in` says that Γ is an MCS in the witnessed model generated by Θ. `Set.is_dummy` says that Γ is the dummy state *and* the dummy state is required in the model. This is how we ensure that, in the words of Blackburn [5], "we only glue on a dummy state when we are forced to".

Chapter 6

Conclusions. Further work

This concludes our investigation of hybrid logic and its application in Lean. We have seen that Lean is expressive enough to allow the statement of all relevant definitions and properties related to our system, and it has also allowed us to prove a large portion of these results. As for the missing results, we see a clear path ahead for the proof of all of them. We conclude our discussion by an outline of this path.

Our implementation relied on the infiniteness of the set NOM of nominal symbols. In the future, work will be needed to allow for languages with finite sets of nominals. It should not be problematic to transfer the results between the two sets of languages, since the finite case is contained in the infinite one. The steps we have outlined in our discussion of language expansions in section 5.2.2 should provide the basis of this task. Work is also still needed to complete the proof of the existence lemma for completed models. We have already formalized the construction needed for this proof, but we are still yet to show that our construction satisfies all the desired properties. As a long-term goal, we may look into generalizing our results to the case of the hybrid logics presented in [22].

For the purposes of this presentation, however, we consider that we have succeeded in offering a general overview on the nature of hybrid logic, and also on the way in which computers can assist formal research in logic. We hope that we have also been able to vindicate our claim from the introduction of this work that *lines of reasoning can have direct analogues in lines of code*.

Bibliography

[1] Jeremy Avigad, Leonardo de Moura, Soonho Kong, and Sebastian Ullrich. *Theorem Proving in Lean 4*. URL: https://leanprover.github.io/theorem_proving_in_lean4/ (visited on 08/16/2023).

[2] Bruno Bentzen. “A Henkin-Style Completeness Proof for the Modal Logic S5”. *Logic and Argumentation*. Ed. by Pietro Baroni, Christoph Benzmüller, and Yi Wáng. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2021, pp. 459–467.

[3] Nicole Bidoit and Dario Colazzo. “Testing XML Constraint Satisfiability”. *Electronic Notes in Theoretical Computer Science*. Proceedings of the International Workshop on Hybrid Logic (HyLo 2006) 174 (2007), no. 6, pp. 45–61.

[4] Patrick Blackburn. “Arthur Prior and Hybrid Logic”. *Synthese* 150 (2006), no. 3. Publisher: Springer, pp. 329–372.

[5] Patrick Blackburn. “Hybrid Completeness”. *Logic Journal of IGPL* 6 (1998), no. 4, pp. 625–650.

[6] Patrick Blackburn, Maarten de Rijke, and Yde Venema. *Modal Logic*. 4. print. with corr. Cambridge Tracts in Theoretical Computer Science 53. Cambridge: Cambridge Univ. Press, 2010. 554 pp.

[7] Luca Cardelli and Giorgio Ghelli. “TQL: A Query Language for Semistructured Data Based on the Ambient Logic”. *Mathematical Structures in Computer Science* 14 (2004), no. 3. Publisher: Cambridge University Press, pp. 285–327.

[8] William Chan. “Temporal-Logic Queries”. *Computer Aided Verification*. Ed. by E. Allen Emerson and Aravinda Prasad Sistla. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer, 2000, pp. 450–463.

[9] Horațiu Cheval and Bogdan Macovei. *Matching Logic in Lean*. GitLab. 2023. URL: https://gitlab.com/ilds/aml-lean/MatchingLogic/-/tree/master/MatchingLogic.

[10] Jan Chomicki, David Toman, and Michael Böhlen. "Querying ATSQL Databases with Temporal Logic". *ACM Transactions on Database Systems* 26 (2001), no. 2, pp. 145–178.

[11] Carlo Combi, Andrea Masini, Barbara Oliboni, and Margherita Zorzi. "A Hybrid Logic for XML Reference Constraints". *Data & Knowledge Engineering* 115 (2018), pp. 94–115.

[12] Carlo Combi, Andrea Masini, Barbara Oliboni, and Margherita Zorzi. "A Logical Framework for XML Reference Specification". *Database and Expert Systems Applications*. Ed. by Qiming Chen, Abdelkader Hameurlain, Farouk Toumani, Roland Wagner, and Hendrik Decker. Lecture Notes in Computer Science. Cham: Springer International Publishing, 2015, pp. 258–267.

[13] Agostino Dovier and Elisa Quintarelli. "Applying Model-Checking to Solve Queries on Semistructured Data". *Computer Languages, Systems & Structures* 35 (2009), no. 2, pp. 143–172.

[14] Herbert Enderton. *A Mathematical Introduction to Logic*. 2nd ed. San Diego: Harcourt/Academic Press, 2001. 317 pp.

[15] Robert Goldblatt. *Logics of Time and Computation*. 2. ed., rev. and expanded. CSLI lecture notes 7. Stanford, CA: Center for the Study of Language and Information, 1992. 180 pp.

[16] Valentin Goranko and Antje Rumberg. "Temporal Logic". In: *The Stanford Encyclopedia of Philosophy*. Ed. by Edward Zalta and Uri Nodelman. Fall 2023. Metaphysics Research Lab, Stanford University, 2023.

[17] Jesse Michael Han and Floris van Doorn. "A Formalization of Forcing and the Unprovability of the Continuum Hypothesis". *10th International Conference on Interactive Theorem Proving (ITP 2019)*. Ed. by John Harrison, John O'Leary, and Andrew Tolmach. Vol. 141. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2019, 19:1–19:19.

[18] David Harel, Dexter Kozen, and Jerzy Tiuryn. *Dynamic Logic*. Foundations of Computing series. Cambridge, Mass. London: MIT Press, 2000. 459 pp.

[19] Leon Henkin. “The Completeness of the First-Order Functional Calculus”. *Journal of Symbolic Logic* 14 (1949), no. 3, pp. 159–166.

[20] William Alvin Howard. “The Formulae-as-Types Notion of Construction”. In: *To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism*. Ed. by Haskell Curry, Roger Hindley, and Jonathan Seldin. Academic Press, 1980.

[21] Stephen Cole Kleene. *Mathematical Logic*. Dover ed. Mineola, N.Y: Dover Publications, 2002. 398 pp.

[22] Ioana Leuștean, Natalia Moangă, and Traian Florin Șerbănuță. “From Hybrid Modal Logic to Matching Logic and Back”. *Electronic Proceedings in Theoretical Computer Science* 303 (2019), pp. 16–31. arXiv: 1907.05029[cs].

[23] Jiatu Li. *Formalization of PAL S5 in Proof Assistant*. Version 1. 2020. arXiv: 2012.09388.

[24] *Mathlib.Data.Countable.Defs*. URL: https://leanprover-community.github.io/mathlib4_docs/Mathlib/Data/Countable/Defs.html (visited on 08/17/2023).

[25] *Mathlib.Data.List.Defs*. URL: https://leanprover-community.github.io/mathlib4_docs/Mathlib/Data/List/Defs.html#List.dedup (visited on 08/18/2023).

[26] *Mathlib.Data.List.Sort*. URL: https://leanprover-community.github.io/mathlib4_docs/Mathlib/Data/List/Sort.html#List.merge (visited on 08/18/2023).

[27] *Mathlib.Data.Nat.Factors*. URL: https://leanprover-community.github.io/mathlib4_docs/Mathlib/Data/Nat/Factors.html#Nat.factors_unique (visited on 08/17/2023).

[28] *Mathlib.Data.Set.Basic*. URL: https://leanprover-community.github.io/mathlib4_docs/Mathlib/Data/Set/Basic.html (visited on 08/17/2023).

[29] *Mathlib.Logic.Denumerable*. URL: https://leanprover-community.github.io/mathlib4_docs/Mathlib/Logic/Denumerable.html (visited on 08/17/2023).

[30] *Mathlib.Logic.Encodable.Basic*. URL: https://leanprover-community.github.io/mathlib4_docs/Mathlib/Logic/Encodable/Basic.html (visited on 08/17/2023).

[31] *Maths in Lean: Sets and set-like objects*. URL: https://leanprover-community.github.io/theories/sets.html (visited on 08/18/2023).

[32] John Ellis McTaggart. "The Unreality of Time". *Mind* 17 (1908), no. 68. Publisher: Oxford University Press, Mind Association, pp. 457–474.

[33] Leonardo de Moura, Soonho Kong, Jeremy Avigad, Floris van Doorn, and Jakob von Raumer. "The Lean Theorem Prover (System Description)". *Automated Deduction - CADE-25*. Ed. by Amy Felty and Aart Middeldorp. Cham: Springer International Publishing, 2015, pp. 378–388.

[34] Andrei-Alexandru Oltean. *Hybrid Logic in Lean*. 2023. URL: https://github.com/alexoltean61/hybrid_logic_lean.

[35] Arthur Norman Prior. *Past, Present and Future*. Oxford: Clarendon Press, 1978.

[36] Grigore Roșu. "Matching Logic". *Logical Methods in Computer Science* Volume 13, Issue 4 (2017). Publisher: Episciences.org.

[37] Grigore Roșu and Traian Florin Șerbănuță. "An Overview of the K Semantic Framework". *The Journal of Logic and Algebraic Programming* 79 (2010), no. 6, pp. 397–434.

[38] *tactic.lean_core_docs - mathlib docs*. mathlib for Lean - API documentation. URL: https://leanprover-community.github.io/mathlib_docs/tactic/lean_core_docs.html (visited on 08/20/2023).

[39] *What is Lean - Lean Manual*. URL: https://leanprover.github.io/lean4/doc/ (visited on 08/20/2023).

[40] Freek Wiedijk. *Formalizing 100 Theorems*. URL: https://www.cs.ru.nl/~freek/100/ (visited on 08/19/2023).