Senior Thesis in Mathematics

Infiniteness and Linear Temporal Logic:

Soundness, Completeness, & Decidability

Author:
Eric Hayden Campbell

Advisor:
Dr. Michael Greenberg

Submitted to Pomona College in Partial Fulfillment
of the Degree of Bachelor of Arts

April 12, 2017,
Minor Revisions: July 8, 2017

Abstract

In this thesis, we present sound and complete axiomatic framework for Linear Temporal Logic over finite traces (LTL_f), as well as a decision procedure. This is founded on the work on LTL of [21, 14, 25, 24], and the specific work on LTL_f in [8, 9, 4]. Our approach follows the least fixpoint method for LTL given in [21], using insights about finitude presented in [25, 24]. Finally, the completeness of LTL_f extends the partial completeness proof for Temporal NetKAT to a full completeness result.

Contents

Chapter 1

Introduction

Temporal logics give us ways to reason about how various objects change over discrete time steps. Temporal reasoning is invaluable for escaping the functional model of program verification, where programs are modelled as functions and verifications are based on the input and output of those functions. However, sometimes we want to analyze the internal behaviour of programs independent of their output, and so an analysis of their correctness under this functional model leaves something to be desired.

As an example, consider the flight plans of two passengers Speedy and Sleepy, both travelling from their homes in San Francisco to a conference in New York. Speedy, a senior professor, is flying direct from SFO to JFK, whereas Sleepy, a student, is flying from SFO to PHX to DFW to JFK. If we a consider a function $\mathsf{ports} : \mathit{Traveler} \to \mathit{Port} \times \mathit{Port}$ that collects the first and last airports for each traveler, we see that $\mathsf{ports}(\text{Sleepy}) = \mathsf{ports}(\text{Speedy}) = (\text{SFO}, \text{JFK})$. Our functional interpretation presents Speedy's and Sleepy's travel as equivalent, however we know that Sleepy probably spent less money, and had a more exhausting experience, whereas Speedy probably paid more money for their ticket, and had a more pleasant experience.

Temporal logic allows us to differentiate Speedy's travel from Sleepy's travel by asking questions like "if the traveler is at SFO, will they be in JFK next?" which is true for Speedy and false for Sleepy. We can also specify ultimate correctness goals like "if the traveler is at SFO, will they ever end their journey in JFK?".

Another use case for temporal logic is the class of functions with side effects. For example, database systems [6], operating systems [6], and Mars Rovers [2, 3] are programs that begin running with little to no input, accepting instead inputs during the run of the program, and and producing outputs *before termination.* So in these situations, a functional model, while providing important and necessary information, is not sufficiently nuanced. To formally verify their systems, then, the above authors use the most widely-used temporal logic, called Linear Temporal Logic, or LTL [28].

Conventionally, LTL's model of time assumes the possibility of infinite time. Recently, researchers have more closely considered a variant of LTL, called LTL$_f$ [8], that assumes a strictly finite model of time, analyzing both the theoretical properties [9, 8] and applying it to network programming to reason about the history of a packet in a network [4].

Before we begin to study LTL$_f$ in earnest, we first need to understand how to formulate and study a *logic*. We understand [18] a logic to be comprised of three parts: a *syntax*, a *semantics*, and a *theory* i.e. an axiomatic framework. We will descibe these in more detail presently. Once we have the entire logic specification, we can begin to prove certain properties about the logic. The important properties that we consider are soundness, completeness and decidability.

Before we jump into reasoning about Temporal Logic (Sections 2 - 3q) we study the the *syntax*, *semantics*, *axiomatic framework*, *soundness*, and *completeness* of the simpler and more familiar context of Classical Logic. Once we complete that we will study the cannonical temporal logic, LTL, and its properties of soundness and completeness. Then, we will present the meat of the paper, a sound and complete axiomatic framing for LTL$_f$ and a decision procedure. We will conclude with an analysis of related work and future directions.

1.1 Syntax

Around the turn of the century, mathematicians and philosophers became involved in self-study, attempting to discover what mathematical reasoning *was*, and if it worked. Of course, this includes problems like "why does $2 + 2 = 4$ and not 5?" and "Does the set of all sets contain itself?", but it really attempts to solve a much more fundamental problem:

How do we know that our proofs prove what we think they do?

For some, this is a rather simple problem. A student will submit a proof, as part of a homework set, to an oracle. The oracle decides whether the proof is correct, and adjusts the student's grade accordingly. But this begs the question! When a mathematician has a *new* proof, they have no oracle to turn to, so she will consult her colleagues, friends, family, and dog, who will decide if they *believe* that the proof works. Then she will submit it to "the community", which will then decide whether it *believes* the proof.

In studying proofs, it is futile to try and model this "belief" approach when analyzing proof in general. It similarly unwise to attempt to study every proof ever written and determine its correctness. To avoid absurdity, we need to come up with effective abstractions for proofs as well as their truth. So, we will first consider the even more fundamental question of *What is a proof?* Typically, a proof is an sequence of sentences designed to convice a reader of the truth of a statement. So, we need to abstract from specific sentences to generic sentences and define way to combine and maniplate the *form* of these sentences in a way that is universally convincing.

To abstract sentences, we need a set of symbols that define sets of these sentences. Of course we can only represent a small subset of these sentences, but we can capture many of the important ones. We will call the set of symbols a *syntax*. The syntax we will start with in Definition 1.1 is the ubiquitous Classical Propositional Logic [5, 18].

Definition 1.1 (Syntax for Classical Logic)**.** A *formula* can be either a variable v from some set $\mathbf{V}$, the symbol $\bot$, pronounced "bot", "bottom", or "false", or an implication, $a \to b$, pronounced "a implies b", where a and b are other formulae. We can define this syntax symbolically below:

$$a, b \ \texttt{::=} \ v \ \mid \ \bot \ \mid \ a \to b$$

The set of formulae that can be generated by the set $\mathbf{V}$ and using the above syntax is $CL(\mathbf{V})$, that is, every formula in $CL(\mathbf{V})$ is $v \in \mathbf{V}$, the symbol $\bot$, or $a \to b$ where $a \in CL(\mathbf{V})$ and $b \in CL(\mathbf{V})$.

This syntax allows us to abstractly represent sentences like "If Socrates is a man, then Socrates is mortal." To do this, we specify the set $\mathbf{V}$ to be

$$\mathbf{V} = \{\text{"Socrates is mortal"}, \text{"Socrates is a man"}\}$$

.

Then we can abstractly represent the sentence as "Socrates is a man" $\to$ "Socrates is mortal". Often, however, we don't care about the content of the sentences, so we will leave the set $\mathbf{V}$ unspecified and only consider arbitrary elements thereof.

This simple implication model seems like it might be limiting. How can we represent that we believe a and b? What about if we believe that a holds or b holds? Or what if we just think that b is false? To represent these kinds of sentences we will use *syntactic sugar*, which is a way of including more operators as a shorthand for more complicated combinations of symbols.

Definition 1.2 (Syntactic Sugar for Classical Logic).

Negation "a doesn't hold"

$$\neg a \triangleq a \to \bot$$

Disjunction "a or b holds"

$$a \vee b \triangleq \neg a \to b$$

Conjunction "a holds and b holds"

$$a \wedge b \triangleq \neg(\neg a \vee \neg b)$$

Lets walk through an intuitive justification for each of these encodings.

Negation We want to express that a does not hold. If a doesn't hold that means we should get a contradiction whenever we try and claim a. The implication goes both ways, so we can encode $\neg a$ as $a \to \bot$.

Disjunction Here we want to express that a holds or b holds. Either a holds or it doesn't. If a does hold, then certainly "a or b" holds. If a doesn't hold, then we say $\neg a$ holds, and since $\neg a \to b$ holds, b must hold and so "a or b" also holds. The same justification works if we swap the terms a and b, so we can equivalently encode $a \vee b$ as $\neg b \to a$.

Conjunction Here we are simply using DeMorgan's laws [10]. If a holds and b holds, then neither $\neg a$ nor $\neg b$ holds. This goes both directions, so we encode $a \wedge b$ as $\neg(\neg a \vee \neg b)$.

Remark 1.3. Notice that using syntactic sugar is a compact way of specifying common behavior. For example if we were to represent $a \wedge b$ in all its unsugared glory, we would have to write

$$a \wedge b \triangleq (a \to (b \to \bot)) \to \bot$$

Remark 1.4. Our chosen basic syntax isn't the only way of encoding Classical logic. We could have also chosen to make $\wedge$ and $\neg$ our simple operators, encoding instead $a \vee b \triangleq \neg(\neg a \wedge \neg b)$ and $a \to b \triangleq \neg a \vee \neg b$.

a	b	$a \to b$
true	**true**	**true**
true	**false**	**false**
false	**true**	**true**
false	**false**	**true**

Figure 1.1: Truth Table for Implication

1.2 Classical Logic: Semantics

So far, we have been thinking about understanding formulae as abstractions of sentences in a proof. In practice, however, we are only concerned about whether those statements are true. For example. If we are trying to reason about sentences of the form "If Socrates is a cat then Darwin is a monkey", we don't care about Socrates or Darwin, rather Socrates' cattiness and whether Darwin really wants a banana. So instead, we will enumerate all possible worlds. These worlds are

Socrates is a cat	Darwin is a monkey
Socrates is a cat	Darwin is NOT a monkey
Socrates is NOT a cat	Darwin is a monkey
Socrates is NOT a cat	Darwin is NOT a monkey

By examining these potential worlds, we can then examine in which cases the statement "Socrates is a cat" $\to$ "Darwin is a monkey" is **true** (for some definition of true). To do this, we need a method of assigning these truth values within one of these possible worlds. For arbitrary statements of the form $a \to b$, we can consider how the truth of a and b affects the truth of the statement $a \to b$. We represent this in a table similar to the one above, called a *truth table*. A truth table is a grid that allows you to quickly perform the recursion inherent in the structure of a formula for all possible values of the variables. We demonstrate an abstracted version of the Socrates/Darwin table of this in Figure 1.1

Truth tables allow for an visualization of how the smaller operators combine into bigger ones. They are also useful for verifying encodings of operators. To do so, simply draw up the two tables and ensure that given the same truth values for the variables, the output is the same. Of course, for an encoding, such as $a \wedge b$, we don't have a definition for how the formula

a	b	$a \wedge b$
true	**true**	**true**
true	**false**	**false**
false	**true**	**false**
false	**false**	**false**

a	b	$\neg b$	$a \to \neg b$	$\neg(a \to \neg b)$
true	**true**	**false**	**false**	**true**
true	**false**	**true**	**true**	**false**
false	**true**	**false**	**true**	**false**
false	**false**	**true**	**true**	**false**

Figure 1.2: Verification of $\wedge$ encoding

behaves. So we have to define how we *want* the operator to behave, and then verify that our encoding behaves appropriately. This is done in Figure 1.2. Notice that the highlighted columns are exactly the same for the two tables.

While truth tables are useful in some situations, we often want to provide a more general approach to reasoning about which possible world we are in. We use a *valuation function* to define a possible world.

Definition 1.5 (Valuation Function)**.** A *valuation function* is defined to be a function $\eta : \mathbf{V} \to \{\textsf{\textbf{true}}, \textsf{\textbf{false}}\}$

Now we can mathematically represent rows in the truth tables. For instance, if we have a function η such that $\eta(\text{“Socrates is a cat”}) = \textsf{\textbf{true}}$, then we only need to consider possible worlds where Darwin is and isn't a monkey. Now, we can use this class of functions to build up meaning for our more complex operators. We will use the function val_η to do this for Classical Logic.

Definition 1.6 (Valuation Function for Classical Logic)**.** A *valuation function for Classical Logic*, written $\mathsf{val}_\eta : \mathrm{CL}(\mathbf{V}) \to \{\textsf{\textbf{true}}, \textsf{\textbf{false}}\}$, is an extension of a valuation fuction η, where $\mathrm{CL}(\mathbf{V})$ is the set of all syntactic terms in Classical Logic over the domain $\mathbf{V}$ as defined in 1.1. The function val_η is applied to the terms in $\mathrm{CL}(\mathbf{V})$ as follows:

($\bot$) $\mathsf{val}_\eta(\bot) = \textsf{\textbf{false}}$

(v) For $v \in \mathbf{V}$, $\mathsf{val}_\eta(v) = \eta(v)$

($\to$) $\mathsf{val}_\eta(a \to b) = \textsf{\textbf{true}}$ if $\mathsf{val}_\eta(a) = \textsf{\textbf{false}}$ or $\mathsf{val}_\eta(b) = \textsf{\textbf{true}}$

Now that we have a functional interpetation, want to define a relation $\vDash$ pronounced “models” that will generalize the valuation function. We will write $\mathcal{F} \vDash a$ when we want to convey that given a valuation function for which a certain set of formulae $\mathcal{F}$ hold, a will certainly hold. This is a kind of metamathematical implication that allows us to say, "if I know that the formulae in $\mathcal{F}$ are true, then I also know a". We define this formally below in Definition 1.7

Definition 1.7 (The Models Relation for Classical Logic)**.** Given a formula a and a set of formulae $\mathcal{F}$, we write $\mathcal{F} \vDash a$ when for all $\eta : CL(\mathbf{V}) \to \{\textsf{\textbf{true}}, \textsf{\textbf{false}}\}$, if for all $b \in \mathcal{F}$, $\mathsf{val}_\eta(b) = \textsf{\textbf{true}}$, then we can show that $\mathsf{val}_\eta(a) = \textsf{\textbf{true}}$.

We say that a formula a is a *consequence* of $\mathcal{F}$, when $\mathcal{F} \vDash a$. If $\mathcal{F} = \emptyset$, then we say that a is *(universally) valid* and write $\vDash a$.

Example 1.8. Lets look at an example to understand how this works. Consider the set of formulae $\mathcal{F} = \{a \to b, a\}$ and the formula $a \wedge b$. We want to see if $a \wedge b$ is a consequence of $\mathcal{F}$. So consider an arbitrary valuation function $\eta : CL(\mathbf{V})$, such that $\mathsf{val}_\eta(a \to b) = \textsf{\textbf{true}}$ and $\mathsf{val}_\eta(a) = \textsf{\textbf{true}}$, and we want to show that in this valuation function $\mathsf{val}_\eta(a \wedge b) = \textsf{\textbf{true}}$. So, we have that $\mathsf{val}_\eta(a) = \textsf{\textbf{false}}$ or $\mathsf{val}_\eta(b) = \textsf{\textbf{true}}$, the first case contradicts our other assumption that $\mathsf{val}_\eta(a) = \textsf{\textbf{true}}$. So we know that $\mathsf{val}_\eta(b) = \textsf{\textbf{true}}$. Since both a and b are mapped to **true**, then we see that $\mathsf{val}_\eta(a \wedge b) = \textsf{\textbf{true}}$.

We have given a semantics for Classical Logic, and discussed several semantic models that allow us to understand what formulae (our abstraction of sentences) *mean* on an intuitive level. However, we still don't know how abstractly reason about them independent of their semantic truth. We still need a solid way to construct proofs of formulae so that we can know which moves are allowable in proofs.

1.3 A Proof System

Now we leave the world of truth, and enter the world of proof. When we consider a semantic interpretation of the formula, we wonder if its valuation returns **true** or **false**. However, in the world of proof we create a set of axioms and inference rules, and wonder if a formula can be derived by application and combination of these axioms and inference rules.

Axioms are formulae that we presume to hold. They are often so "obvious" or simple that any rational viewer should be easily able to ascertain their correctness. Sometimes, this means doing complicated math! We represent axioms and inference rules in the standard way [5]. For an arbitrary rule, we write

$$\frac{\langle\text{Premises}\rangle}{\langle\text{Conclusion}\rangle} \tag{\textsc{NameOfRule}}$$

where the premises are a set of formulae and the conclusion is a single formula. By this notation, we means that the formula $\bigwedge_{p \text{ a Premises}} f$ implies the Conclusion. there are no assumptions, the rule is called an *axiom*. Otherwise, it is an *inference rule*.

We will also define a relation $\cdot \vdash \cdot \subseteq 2^{CL(V)} \times CL(\mathbf{V})$ pronounced "proves". Writing $\mathcal{F} \vdash a$ means that according to a set of rules, a can be derived from the set of assumptions $\mathcal{F}$. We will write $\vdash$ for $\emptyset \vdash a$.

The axioms, inference rules, and $\vdash$ relation, allow us to reason about what formulae are *provable*. Let's see how these pieces fit together with respect to Classical Logic. First we will define the proof system for classical logic.

Definition 1.9 (Proof System for Classical Logic)**.** Let the following axioms and inference rules specify the allowable terms in the $\vdash$ relation.

The axioms for Classical Logic are:

$$\vdash a \to a \tag{\textsc{Id}}$$

$$\vdash a \to (b \to a) \tag{\textsc{Weak}}$$

$$\vdash (a \to (b \to c)) \to ((a \to b) \to (a \to c)) \tag{\textsc{Distr}}$$

$$\vdash (\neg b \to \neg a) \to (a \to b) \tag{\textsc{ContraPos}}$$

The single inference rule for Classical Logic is

$$\frac{\vdash a \to b \qquad \vdash a}{\vdash b} \tag{\textsc{ModusPonens}}$$

To better understand and believe these rules, let's show their truth tables. For each of these rules to be sensible, we expect them to be universally valid, i.e. true in *all possible worlds*. This means that the column containing the rule should be entirely **true**.

a	$a \to a$
true	**true**
false	**false**

Figure 1.3: Truth Table for Id

a	b	$b \to a$	$a \to (b \to a)$
true	**true**	**true**	**true**
true	**false**	**true**	**true**
false	**true**	**false**	**true**
false	**false**	**true**	**true**

Figure 1.4: Truth Table for Weak

The most interesting case is when we consider ModusPonens rule. This means that we do not consider all possible worlds, we only consider those worlds in which a is true and $a \to b$ is true. So, in the truth table, we consider only rows where $a \mapsto$ **true** and $b \mapsto$ **true** (Figure 1.7).

1.4 Soundness and Completeness

We have an axiomatic framework and a mathematical semantic interpretation of formulae in the logic; are they really talking about the same thing? That is, do our notions of proof and truth agree?

Informally, *soundness* is the claim the the axiomatic framework doesn't allow us to prove claims that are not "true" in the intuitive semantic interpretation (whatever "true" means [26, 18]). This serves as a kind of sanity check; the rules we've defined to govern our system actually work. This is critical for any axiomatic system. If the rules are not sound, then the system can derive formulae that are "false" in the model.

Once the rules are proven to be sound, we want to know something about what portion of the set of "true" claims our rules can derive. *Completeness* says that the set of "true" claims is a subset of those things that the rules can derive, meaning that, if a claim is "true", then it can be proven by the axiomatic framework. We define these more formally below:

a	b	c	$b \to c$	$a \to b$	$a \to c$	$a \to (b \to c)$	$(a \to b) \to (a \to c)$	Distr
true	**true**	**true**	**true**	**true**	**true**	**true**	**true**	**true**
true	**true**	**false**	**false**	**true**	**false**	**false**	**false**	**true**
true	**false**	**true**	**true**	**false**	**true**	**true**	**true**	**true**
true	**false**	**false**	**true**	**false**	**false**	**true**	**true**	**true**
false	**true**	**true**	**true**	**true**	**true**	**true**	**true**	**true**
false	**true**	**false**	**false**	**true**	**true**	**true**	**true**	**true**
false	**false**	**true**	**true**	**true**	**true**	**true**	**true**	**true**
false	**false**	**true**	**true**	**true**	**true**	**true**	**true**	**true**

Figure 1.5: Truth table for Distr Axiom

a	b	$\neg a$	$\neg b$	$a \to b$	$\neg a \to \neg b$	ContraPos
true	**true**	**false**	**false**	**true**	**true**	**true**
true	**false**	**false**	**true**	**false**	**true**	**true**
false	**true**	**true**	**false**	**true**	**false**	**true**
false	**false**	**true**	**true**	**true**	**true**	**true**

Figure 1.6: Truth Table for ContraPos Axiom

Definition 1.10 (Soundness)**.** A logic is called *sound* if for every $\mathcal{F}$ a set of formulae in the logic, and a a formula in the logic, $\mathcal{F} \vdash a$ implies $\mathcal{F} \vDash a$.

Definition 1.11 (Completeness)**.** A logic is called *complete* if for $\mathcal{F}$ a set of formulae in the logic and a a formula in the logic, then $\mathcal{F} \vDash a$ implies $\mathcal{F} \vdash a$.

Soundness is a simple property at says that our proofs correspond to reality. It's proof is a sensibility argument that amounts to showing that all of the axioms and inference rules are valid under the model, since every proof is based on some combination of these axioms and inference rules.

In general, it is a very bad if a proof system or logic is not sound, since that would allow you to prove things that are false. Since the proof system is intended to be a formal way of evaluating whether something is true or not (according to some set of rules) then it defeats the purpose if the system is not sound.

In contrast, completeness makes a totality argument, saying that the current set of axioms can prove *every true statement* in the logic. This is not necessary for a logic, and there are many interesting logics that are sound but not complete, in which we can prove interesting things. For instance,

a	b	$a \to b$	b
true	**true**	**true**	**true**
true	**false**	**false**	
false	**true**	**true**	
false	**false**	**true**	

Figure 1.7: Truth Table for the ModusPonens Inference rule

any system of logic or mathematics that has a notion of infinite arithmetic is not complete [15, 16]. So, this would imply that much of mathematics, including fields like Number Theory and Analysis, are incomplete. However, these are quite vibrant fields, in which mathematicians have proven and continue to prove interesting and useful theorems. So completeness doesn't say anything about the "usefulness" or breadth of a system, it is only making a claim about how tight the correspondence between the rules and the concrete interpretation of the symbols.

Proving completeness is typically much harder than proving soundness. To do the latter, we simply need to show that the axioms and inference rules are sensible with respect to some concrete model. To prove completeness, we need to consider an arbitrary formula that is true in our model and show that there exists a proof of that formula.

Let's start by showing that our choice of axioms are sensible, i.e. proving that every axiom and inference rule is true under our semantic model.

Theorem 1.12 (Soundness of Classical Logic)**.** *Classical Logic is sound.*

Proof. We already showed that each of the rules is valid! Examining the truth tables presented in Section 1.3 demonstrates that each of the rules is valid in all possible worlds. This is a combinatorial proof that enumerates all possible inputs and outputs to val_η over all η. △

Theorem 1.13 (Completeness of Classical Logic)**.** *Classical Logic is complete.*

Proof. Given a proposition a such that $\mathcal{F} \vDash a$, show that $\vdash a$. Let $V \subseteq \mathbf{V}$ be the set of variables in $\mathcal{F}$. Consider an arbitrary η function, and let $S_\eta \subseteq CL(\mathbf{V})$ be

$$S = \begin{cases} v, & \text{if } v \in V, \text{ and } \eta(v) = \textsf{\textbf{true}} \\ \neg v, & \text{if } v \in V, \text{ and } \eta(v) = \textsf{\textbf{false}} \end{cases}$$

Notice now that for every $s \in S_\eta$, $\mathsf{val}_\eta(s) = \textbf{true}$. Let $\mathcal{F}' = \{f \vee \neg f \mid f \in \mathcal{F}\}$. Kleene [18] proves a lemma to show that for an arbitrary formula f, then $\mathsf{val}_\eta(f) = \textbf{true}$ if and only if $S \vdash f$. He also shows that for an arbitrary formula f, if for every η, if $S_\eta \vdash f$, then $\mathcal{F}' \vdash f$.

Chaining these together, we can conclude that $\mathcal{F}' \vdash a$. We can now show that for every formula f, we have $\vdash f \vee \neg f$, by desugaring the $\vee$ operator to see $\vdash \neg f \rightarrow \neg f$ which it a tautology by Id. Since every element of $\mathcal{F}'$ is of the form $f \vee \neg f$ we can conclude that $\vdash a$. For more detail, see [18]. $\triangle$

Chapter 2

Linear Temporal Logic

A computer does exactly what a user or programmer tells it to do. This is a blessing and a curse, as being perfectly precise in the instructions one gives is a very difficult thing to do, and a programmer will often vociferate in anguish “That’s what I said, but not what I meant!” Since computers are frustratingly pedantic, we must find a way to *prove* that programs do what we want them to do.

To verify the correctness of programs, computer scientists frequently use constructive, classical, and first-order logic. These are often powerful tools used to prove functional properties for programs that have clearly defined inputs and deliver an output on termination. Sometimes we want to verify properties of programs that are not related to the inputs and outputs and not dependent on termination; programs like operating systems [6], databases [6], or Java execution traces [27]. Linear Temporal Logic (LTL) gives us more power to verify these types of programs.

LTL offers a way to ask questions about the execution of a program [28]. For example, we can use Temporal Logic to examine the predicament of our travellers Sleepy and Speedy who want to get from LAX to JFK. If we treat Sleepy’s flight plan to get from SFO to JFK as a program, we want to make sure that Sleepy *eventually* gets to JFK, that the final state of the program is JFK. If, for some reason, Sleepy has a criminal record in Texas, we can also make sure that they *never* fly though DFW.

To formalize these kinds of statements in LTL, we will use that syntax standard to modal logics, adding the $\square$ (global), and $\bigcirc$ (next) operators, to the standard classical logic operators.

Definition 2.1. For a set of variables $\mathbf{V}$, with $v \in \mathbf{V}$ and a and b propositions, this gives a minimal syntax

$$a, b \ ::= \ v \ \mid \ \bot \ \mid \ \bigcirc a \ \mid \ a \rightarrow b \ \mid \ a \,\mathcal{W}\, b$$

Define $LTL(\mathbf{V})$ to be the set of formulae generated by the above syntax over $\mathbf{V}$. An arbitrary formula $f \in LTL(\mathbf{V})$ is the symbol $\bot$, a variable in $\mathbf{V}$, of the form $\bigcirc a$ where $a \in LTL(\mathbf{V})$, of the form $a \rightarrow b$ where $a, b \in LTL(\mathbf{V})$, or of the form $a \,\mathcal{W}\, b$, where $a, b \in LTL(\mathbf{V})$.

The statement $a \,\mathcal{W}\, b$ intuitively corresponds to the statement "a holds until b holds, or a always does", and $\bigcirc a$ corresponds to the statement "in the next time step, a". For example, to make sure that Sleepy avoids getting arrested due to the warrant for their arrest in Dallas, we can make sure that (Sleepy at SFO) $\rightarrow \neg$(Sleepy at DFW) $\mathcal{W} \bot$, i.e. if Sleepy is at SFO, then Sleepy won't arrive at DFW until $\bot$ holds. Since $\bot$ will never hold, Sleepy will never be at DFW.

Extending this basic syntax, we get the standard classical logic encodings from Definition 1.2 of $\wedge, \vee, \neg$, and $\top$. Further, we can define a couple of pieces of temporal syntactic sugar. The "always a" operator, written as $\Box\, a$, says "now and for every in the future a". The "ever" operator is written $\Diamond\, a$, meaning that "at some point in the future, a." Continuing with our flight-plan example, it is now easy to determine whether Sleepy actually gets to JFK airport via the formula Sleepy at SFO $\rightarrow \Diamond$(Sleepy at JFK), which means that Sleepy being in SFO means that they will eventually get to JFK. Of course, these are just intuitive justifications for these encodings, but we can make these intuitions more concrete with a semantic interpretation of the model, using a Kripke Structure [19].

Definition 2.2. Let $\mathbf{V}$ be a set of variables. Define a *Kripke structure* K as an infinite sequence of valuation functions

$$K = (\eta_1, \eta_2, \eta_3, \ldots)$$

where $\eta_i : \mathbf{V} \rightarrow \{\textsf{true}, \textsf{false}\}$ is a function from variables to truth values. We also define the *temporal valuation function* $K_i : LTL(\mathbf{V}) \rightarrow \{\textsf{true}, \textsf{false}\}$, for some index i of this kripke structure K, to be

$$\begin{aligned}
K_i(v) &= \eta_i(v), \text{ for } v \in \mathbf{V} \\
K_i(\bot) &= \textsf{false} \\
K_i(a \to b) &= \textsf{true} \text{ iff } K_i(a) = \textsf{false}, \text{ or } K_i(b) = \textsf{true} \\
K_i(\bigcirc a) &= \textsf{true}, \text{ or } K_{i+1}(a) = \textsf{true} \\
K_i(a \,\mathcal{W}\, b) &= \textsf{true} \text{ iff there exists } k \geq i \text{ such that } K_k(b) = \textsf{true}, \\
&\qquad \text{and for } i \leq j \leq k, K_j(a) = \textsf{true}, \\
&\qquad \text{or for every } k \geq i, K_k(a) = \textsf{true}.
\end{aligned}$$

We can now set up Sleepy's travel plans using this formalism. We define $K = (\eta_1, \eta_2, \eta_3, \eta_4, \ldots)$, where

$$\eta_1(x) = \begin{cases} \textsf{true} & x = (\text{Sleepy at SFO}) \\ \textsf{false} & \text{otherwise} \end{cases}$$
$$\eta_2(x) = \begin{cases} \textsf{true} & x = (\text{Sleepy at PHX}) \\ \textsf{false} & \text{otherwise} \end{cases}$$
$$\eta_3(x) = \begin{cases} \textsf{true} & x = (\text{Sleepy at PHL}) \\ \textsf{false} & \text{otherwise} \end{cases}$$
$$\eta_4(x) = \begin{cases} \textsf{true} & x = (\text{Sleepy at JFK}) \\ \textsf{false} & \text{otherwise} \end{cases}$$
$$\eta_i(x) = \eta_4(x) \quad \forall i > 4$$

So, then we can evaluate our correctness criterion, that

$$(\text{Sleepy at SFO}) \to \Diamond(\text{Sleepy at JFK}).$$

We start at time step 1, since at every other time step i, it is the case that $\eta_i(\text{Sleepy at SFO}) = \textsf{false}$. So, let's evaluate $K_1(\text{Sleepy at SFO} \to \Diamond(\text{Sleepy at JFK}))$. By definition, this is true if $K_1(\neg(\text{Sleepy at SFO})) = \textsf{true}$ or $K_1(\Diamond(\text{Sleepy at JFK}) = \textsf{true}$. We can simplify the first case to $K_1(\text{Sleepy at SFO}) = \textsf{false}$, but, we know that $\eta_1(\text{Sleepy at SFO}) = \textsf{true}$, so we conclude that $K_1(\neg(\text{Sleepy at SFO})) = \textsf{false}$, so we must be in the other case. Now, we can simplify $K_1(\Diamond(\text{Sleepy at JFK})) = \textsf{\textbf{true}}$ to the new expression $K_1(\Box\,\neg(\text{Sleepy at JFK})) = \textsf{\textbf{false}}$. So, we must find some $i \geq 1$ such that $K_i(\text{Sleepy at JFK}) = \eta_i(\text{Sleepy at JFK}) = \textsf{\textbf{true}}$. This is true when $i = 4$, so conclude that $K_1(\Diamond(\text{Sleepy at SFO}) = \textsf{\textbf{true}}$, and ultimately that $K_1(\text{Sleepy at SFO} \to \Diamond(\text{Sleepy at JFK})) = \textsf{\textbf{true}}$.

Remark 2.3. So far, we have been referring to a "time step" without much justification for what that means. In this model, we mean a time step i to be the world η_i.

Once we have a logic's syntax and its mathematical semantics (as we do now), we can consider the appropriate axiomatic framework, followed by soundness and completeness results. But first, we will asses some of the assumptions made by LTL and show real-world examples of how it is used.

2.1 Background for Linear Temporal Logic

First we present the main assumptions of LTL, and then motivate their usefulness with applications of LTL to the real world verification tools Java Pathfinder [27] and EAGLE [2].

2.1.1 Assumptions

Let's first address the assumptions being made in linear temporal logic. There are three: (1) time is discretizable, (2) there is always a "next time", and (3) if a time step has a "next time" it is unique [21]. The sense of these assumptions is not entirely apparent, so we will take some time to examine and justify each of them.

Time is discretizable Physics understands time as continuous, however, we have presented LTL with a discrete model of time. Why can we do this? In day to day life, we already use a discrete model of time where we say that something occurs *at* time t if it happens between time t and time $t + 1$. Computers take a similar approach, operating under a discrete time model [22]. Since LTL is most frequently used for verifying computer programs, this assumption makes sense for LTL.

There is always a "next time" This is a fairly reasonable assumption, especially when we consider programs like operating systems, and database systems that ought to run indefinitely, and then when they are terminated, correctness is suddenly no longer an issue. In certain situations, however, we may want to want to reason about the end of time. The assumption that there is always a next time is no longer viable. A simpler system forces time to end, adding the assumption that "eventually, time ends." This system is called Linear Temporal Logic over finite traces 3, also known as (LTL$_f$). Another variant of this LTL that questions this assumption is called Past-time LTL, which in addition to introducing operators about the past, allows time to either be finite or infinite.

Uniqueness Here we assume that every time step has a unique successor. With a human, macroscopic understanding of physical processes, it seems obvious that in the next time step there are not two different valid worlds (represented as η-functions). This intuition holds for sequential processes, because there is one sequence of commands that are executed. However, the uniqueness assumption is often not sufficient for concurrent or parallel processes [23], as each thread/process has its own distinct successor. When we throw away the uniqueness assumption, we get a logic called Linear Dynamic Logic (LDL), which, while being extremely powerful, is not as popular as LTL. The uniqueness assumption permits a tractable, easy-to-use formalization.

Once we have these assumptions, one might wonder when these assumptions are good ones to make and how powerful the system of LTL is under these assumptions. The best way to assess the power of a logic is by example; so what follow are summaries of some of the theoretical properties and uses in model-checking software.

2.1.2 Applications

Now that we've examined the assumptions, it is useful to justify them by showing they can be applied to real-world situations. What follows is a description of LTL's importance to hardware verification and of two distinct verification tools that use LTL.

Intel Hardware Verification LTL, along with its variant CTL (computation tree logic), is an essential tool in hardware verification [20, 17, 11]. LTL has been one of the major tools used by Intel [11], used to verify CPU designs for the Intel Pentium 4 processor [30]. Specifically, they use LTL to verify the correctness in the "pre-silicon" stage, i.e. to eliminate bugs in the design of the processor. Following the "pre-silicon" stage is, of course, the "silicon" stage, in which Intel uses standard test-driven bug-finding techniques. They were able to use LTL to verify floating-point properties and arithmetic [13] using a model-checking tool [13] based on LTL that was developed at Intel.

Java Pathfinder Java is an extremely well-known and popular language, and Java Pathfinder [27], a tool developed by the NASA Ames Research Center, is the self-titled "swiss army knife of Java verification". JPF is a virtual machine that runs automatic tests on your code, theoretically trying all possible executions of the program to try and expose vulnerabilities.

Recently, an LTL extension called `jpf-ltl` was added to Java Pathfinder [27]. It allows for the user to specify various LTL formulae about their Java program `jpf-ltl` will attempt to falsify the claim. Most interestingly, a user can specify conditions about sequential and concurrent programs, referring to anything from method calls to program variables.

For example, a user could ensure that if you call a certain method, it behaves properly. Let `foo` be the name a function that computes the action f on its inputs. We want to know that $f(\bar{\mathtt{x}})$, happens eventually (after some latency period). So we could ask `jpf-ltl` to verify the claim

$$\forall \bar{\mathtt{x}}.\, \Box(\mathtt{foo}(\bar{\mathtt{x}}) \to \Diamond\, f(\bar{\mathtt{x}}))$$

which says that it is always the case that, the effect of `foo` on $\bar{\mathtt{x}}$ will eventually occur. Or equivalently, that `foo` terminates.

Mars Rover Another project out of NASA Ames Laboratory is the EAGLE verification tool used to verify event controllers in an experimental NASA rover [2]. EAGLE is based on the observation that the formulae $\Box F$ can be expressed as maximal solution to the equation $X = F \wedge \bigcirc X$ and that $\Diamond F$ is the minimal solution to the equation $X = F \vee \bigcirc X$. Using this notion of minimal and maximal fixpoints, the authors define further a semantics, calculus, and evaluation algorithm.

After its specification and implementation, EAGLE was then applied to test a planetary rover controller called K9. The rover controller was an extensive multi-threaded shared-memory C++ application that needed to be responsive to external stimuli. The verification tool, called X9 (since it explores K9) generated test cases and produced an execution trace for those cases that failed. The paper cites an error detected by X9 having to do with temporally dependent processes terminating improperly. More specifically, for pairs of processes p_1 and p_2, where p_2 depends on the completion of p_1, X9 detected a class of bugs caused by p_2 trying to start before p_1 had terminated.

These real-world examples demonstrate the usefulness of temporal logic in general, but specifically the formal and operational models provided by LTL. They also justify assumptions that underlie LTL and begin to explain the wide acceptance and common usage of LTL.

2.2 Classical Formulation

This section will present the Classical formulation (semantics and proof theory) of LTL and a proof of soundness and completeness. The proof follows a modifed version of the Henkin-Hasenjaeger proof of LTL of completeness for classical logic [21]. The first thing we need here is a set of axioms; the axioms for classical logic and those for predicate logic are remarkably similar.

Definition 2.4 (Proof System for LTL). Define the relation $\cdot \vdash \cdot \subseteq 2^{LTL(\mathbf{V})} \times LTL(\mathbf{V})$ to be the formulae generated by the following axioms and inference rules:

Axioms

$$\vdash \text{all tautologies in Classical Logic} \qquad (\textsc{Taut})$$

$$\vdash \neg \bigcirc a \leftrightarrow \bigcirc \neg a \qquad (\textsc{NegNextDistr})$$

$$\vdash \bigcirc(a \to b) \to \bigcirc a \to \bigcirc b \qquad (\textsc{NextDistr})$$

$$\vdash a \mathcal{W} b \to b \wedge b \vee a \wedge \bullet\, a \mathcal{W} b \qquad (\textsc{WkUntilUnroll})$$

Inference Rules

$$\frac{\vdash a \quad \vdash a \rightarrow b}{\vdash b} \qquad (\textsc{ModusPonens})$$

$$\frac{\vdash a}{\vdash \bigcirc a} \qquad (\textsc{Next})$$

$$\frac{\vdash a \rightarrow b \quad a \rightarrow \bigcirc a}{\vdash a \rightarrow \Box b} \qquad (\textsc{Induction})$$

Definition 2.4 defines the set of axioms for LTL. The first is the TAUT axiom, in which we simply assume that anything that is a tautology in Classical Logic (provable from no assumptions) is an axiom of LTL. The next axiom, NEGNEXTDISTR, simply states that $(\neg\cdot)$ and $(\bigcirc\cdot)$ are distributive. Then ALWUNROLL says that if you have an $\Box a$, then you get a now, and $\Box a$ in the next time step. Then we have MODUSPONENS which is exactly the same as in classical logic, followed by the NEXT rule which says that a proof of a in general, i.e. a proof of a's *validity*, is sufficient to say that in the next time step a holds true, because valid formulae always hold. The final rule is INDUCTION, which says that if $\vdash a \rightarrow b$ and $\vdash a \rightarrow \bigcirc a$ then $\vdash a \rightarrow \Box b$.

As a sanity-check for the rules in Definition 2.4, we state the following soundness result (Theorem 2.8), which says that the proof rules above only admit proofs of valid formulae.

Definition 2.5 (Valid)**.** A formula a of $LTL(\mathbf{V})$ is called *valid* in the temporal structure K for $\mathbf{V}$, denoted $\vDash_K a$, if $K_i(a) = \top$ for every $i \in \mathbb{N}$.

Definition 2.6 (Consequence)**.** For a set of formula $\mathcal{F} \in LTL(\mathbf{V})$, we write $\mathcal{F} \vDash a$, pronounced "a is a consequence of $\mathcal{F}$", if

$$\forall K, (\forall c \in \mathcal{F}, \vDash_K c) \implies \vDash_K b$$

Definition 2.7 (Universal Validity)**.** A formula a is called *universally valid* if a is a consequence the empty set of formulae. This is written $\emptyset \vDash a$, or equivalently $\vDash a$.

Theorem 2.8 (Soundness)**.** *If $\mathcal{F} \vdash a$, then $\mathcal{F} \vDash a$.*

Proof Idea. Induction on the derivation of $\mathcal{F} \vdash a$. [1] $\triangle$

[1] For clarity, we use the $\triangle$ symbol to represent the end of a proof since the $\Box$ symbol forms part of the object of study.

Now we want to assess completeness for LTL. Completeness says that if a formula a is a consequence of the set of formulae $\mathcal{F}$, then, if we assume the formulae in $\mathcal{F}$, we can prove a. The proof relies on several important theorems and lemmata. We will start with a couple of deduction theorems for LTL, which will allow us to convert consequence and provability statements of the form $\mathcal{F} \vdash a$ or $\mathcal{F} \vDash a$, to statements of the form $\vdash b$ and $\vDash c$.

Theorem 2.9 (Semantic Deduction)**.** *For $\mathcal{F}$, a set of formulae, with a and b formulae, then $\mathcal{F}, a \vDash b$ if and only if $\mathcal{F} \vDash \Box\, a \to b$.*

Proof. The proof proceeds by induction on the size of $\mathcal{F}$. △

Theorem 2.10 (Deduction)**.** *For $\mathcal{F}$, a set of formulae, with a and b formulae, then $\mathcal{F}, a \vdash b$ if and only if $\mathcal{F} \vdash \Box\, a \to b$.*

Proof. Each direction is proved separately and directly. △

These smaller results allow for us to convert the problem

$$\mathcal{F} \vDash a \implies \mathcal{F} \vdash a$$

to a simpler one. We can let $\mathcal{F} = a_1, a_2, \dots a_n$ and simplify the problem to

$$\vdash \Box\, a_1 \to \Box a_2 \to \cdots \Box\, a_n \to a$$
$$\implies \vDash \Box\, a_1 \to \Box\, a_2 \to \cdots \Box\, a_n \to a$$

In fact we can generalize this claim even further to

$$\vDash b \implies \vdash b$$

This is a much more manageable claim to prove. Next, we will prove an unsatisfiability result that demonstrates how to bridge the gap between the $\vDash$ and the $\vdash$ relations. Specifically, we will come up with a way to generate a formula b such that $\vdash b$ in the theory implies b is satisfiable in the model. We will use the contrapositive of this result in the completeness proof. To prove this satisfiability theorem, we will introduce a positive-negative pair (PNP) structure.

The intuition behind the notation and terminology is that a PNP is a pair of two sets, a positive set ($\mathcal{F}^+$), which only contains formulae that are provable, and the negative set ($\mathcal{F}^-$), which only contains formulae whose negations are provable. PNPs give us a way to separate sets of formulae into those that we think are true and those that we think are false. We enforce with a *consistency* property that allows us to turn our beliefs about truth to conclusions about provability. These are defined formally below

Definition 2.11 (Positive-Negative Pair)**.** A *Positive Negative Pair* (PNP) is a pair of sets of formulae $(\mathcal{F}^+, \mathcal{F}^-)$. For a PNP $\mathcal{P}$, We define the *literal interpretation of* $\mathcal{P}$, $\widehat{\mathcal{P}}$ to be the formula:

$$\widehat{\mathcal{P}} \triangleq \bigwedge_{a \in \mathcal{F}^+} a \wedge \bigwedge_{b \in \mathcal{F}^-} \neg b$$

A PNP $\mathcal{P}$ is *inconsistent* if $\vdash \neg\widehat{\mathcal{P}}$, and *consistent* otherwise. Let the set of all PNPs be *PNP*.

Now we can prove the Satisfiability Theorem (Theorem 2.12), which says that the literal interpretation of a consistent PNP is satisfiable in the model. Here we are relating a statement about provability ("a PNP P is consistent") to one about the the model ("there is a model K_i for which $K_i(\widehat{\mathcal{P}}) = \textbf{true}$").

Theorem 2.12 (Satisfiability for LTL)**.** *For $\mathcal{P}$, a consistent PNP, the formula $\widehat{\mathcal{P}}$ is satisfiable.*

Proof idea. Evaluate the way that the PNPs change throughout time (defining a transition function σ in the full proof) to ensure that each of the temporal operators ($\bigcirc$ and $\square$) behave appropriately. Then construct a Kripke structure K based on the evolution of the PNP throughout time, and show that $K_0(\widehat{\mathcal{P}}) = \textbf{true}$. $\triangle$

Theorem 2.13 (Completeness for LTL)**.** *For $\mathcal{F}$ a set of formulae, and a a formula, if $\mathcal{F} \vDash a$, then $\mathcal{F} \vdash a$.*

Proof Idea. Assume that $\mathcal{F} \vDash a$, where $\mathcal{F} = a_1, \ldots, a_n$. Using Theorem 2.9, we get $\vDash \square\, a_1 \to \cdots \to a_n \to a$. We can perform an analogous transformation on our desired result, using Theorem 2.10. So that we need to show

$$\vdash \square\, a_1 \to \cdots \to \square\, a_n \to a.$$

Broaden the claim so that for all formulae b, $\vDash b$ implies $\vdash b$. Let the PNP $\mathcal{P} = (\emptyset, \{b\})$. Notice that $\widehat{\mathcal{P}} \equiv \neg b$. Then by the contrapositive of Theorem 2.12, we see $\vdash \neg\neg b$, which then by TAUT derives $\vdash b$. $\triangle$

Chapter 3

Linear Temporal Logic over finite traces

We present our main contribution: a presentation and axiomatization of Linear Temporal Logic over finite traces (LTL$_f$), highlighting explicit differences between itself and standard Linear Temporal Logic (LTL). Importantly, we derive soundness (Theorem 3.20), and Completeness (Theorem 3.43). We conclude with a novel decision procedure and a corresponding proof that satisfiability is decidable (Theorem 3.58).

While the core syntax of LTL$_f$ will remain the same as that of LTL, we introduce some new pieces of syntactic sugar, namely end, pronounced "end", and $\bullet\, a$, pronounced "weak next a". The symbol end means that there is no next state, and $\bullet\, a$ means either $\bigcirc\, a$ or end, so if a holds in every state, $\bullet\, a$ will also be true in every state, whereas $\bigcirc\, a$ will be false in the final state.

We also adjust the semantics to assert that time will certainly end. In the underlying model, we will still use Kripke structures from LTL to represent the evolution of the world through time. In our finite setting, we define a finite variant of these Kripke structures with exactly n states. When we're in this nth and last state, and we try and make a statement like $\bigcirc\, a$, which asks to evaluate a in the $n+1$th state, then we immediately evaluate to $\textsf{\textbf{false}}$, without ever looking at how a evaluates.

Similarly, in the proof theory, we will include the axiom FINITE, which is $\vdash \Diamond\, \mathsf{end}$, and says that eventually, time ends. When time ends, we want to mimic the behavior of the semantics, i.e. that trying to make a statement of the form $\bigcirc\, a$ is contradictory. To incorporate this behaviour, we also include the axiom ENDNEXTCONTRA, $\vdash \mathsf{end} \to \neg \bigcirc\, a$, meaning that if we are in the last state, $\bigcirc a$ is contradictory for all formulae a. Further, to compensate for the end of time, we replace all instances of $\bigcirc a$ with $\bullet a$ in the unrolling, step, and induction rules. Notice that, in LTL, the formula $\Diamond \mathsf{end}$ is *unsatisfiable.*

Finally, in the proofs of soundness, completeness, and decidability, we will use a similar method to the proofs of those properties for LTL [21], taking special care to account for these changed assumptions.

3.1 Syntax of LTL$_f$

The syntax for LTL$_f$ is th same as LTL's (Definition 2.1).

$$a, b \;::=\; \bigcirc a \;\mid\; a \,\mathcal{W}\, b \;\mid\; a \rightarrow b \;\mid\; \bot \;\mid\; v$$

Figure 3.1: Syntax for LTL$_f$($\mathbf{V}$)

Our syntax subsumes that of Classical Logic (Chapter 1), namely the binary implication operator $\rightarrow$, the constant $\bot$, and the ability to use variables from the alphabet $\mathbf{V}$. From each of these we can encode the more widely recognized operators, $\wedge$, $\vee$, $\neg$, $\top$, as presented in Definition 1.2.

We also have the temporal operators from LTL, the *weak until* operator $\mathcal{W}$, and the *next* operator $\bigcirc$. Intuitively, a statement $a \,\mathcal{W}\, b$ can be thought of as "moving forward in time, a holds until b holds, or a always holds". Similarly the statement $\bigcirc a$ can be thought of as "in the next time step a holds true".

Recall from Section 2 that unary operators bind tighter than binary operators, so $\Box$, $\bigcirc$, and $\neg$ bind tighter than $\rightarrow$, $\wedge$, and $\vee$, and $\wedge$ binds more tightly than $\vee$ which binds more tightly than $\rightarrow$. As an example, the formula $\bigcirc a \rightarrow \Box d \wedge c$ is equivalent to $(\bigcirc a) \rightarrow ((\Box d) \wedge (\neg c))$.

3.2 Semantics for LTL$_f$

In LTL, the underlying model is an infinite Kripke structure [19], defined as an infinite tuple of functions that assign truth values to the variables. In contrast, for LTL$_f$, we use a finite version, forcing the underlying tuple to have only finitely many functions.

Definition 3.1 (Finite Kripke Structure)**.** A *finite Kripke structure* K^n is an n-tuple of functions

$$K^n = (\eta_1, \eta_2, \ldots, \eta_n)$$

where η_i is a function $\eta_i : \mathbf{V} \to \{\textsf{true}, \textsf{false}\}$.

To give a semantics to the formulae generated by the grammar, we first define a function K_i^n interpreting a formula at time-step i, defined as a fixpoint on the formula. Different from LTL, we need to handle what happens in the last state. In our finite setting, we make sure that any occurence of $\bigcirc\, a$ at the nth always give **false**, without ever looking at a.

Definition 3.2. Given a finite Kripke structure $K^n = (\eta_1, \cdots, \eta_n)$, we define the *temporal valuation function* K_i^n at state i as a fixpoint on its input:

$$\begin{aligned}
K_i^n(v) &= \eta_i(v)\\
K_i^n(\bot) &= \textsf{false}\\
K_i^n(a \to b) &= \begin{cases} \textsf{true} & \text{if } K_i^n(a) = \textsf{false}\\ \textsf{true} & \text{if } K_i^n(b) = \textsf{true}\\ \textsf{false} & \text{otherwise}\end{cases}\\
K_i^n(\bigcirc\, a) &= \begin{cases} K_{i+1}^n(a) & \text{if } i < n\\ \textsf{false} & \text{otherwise}\end{cases}\\
K_i^n(a \,\mathcal{W}\, b) &= \begin{cases} \textsf{true} & \text{if } K_j^n(a) = \textsf{true},\ \text{for all } i \le j \le n\\ \textsf{true} & \text{if there exists } i \le k \le n,\ \text{such that } K_k^n(b) = \textsf{true}\\ & \quad \text{and for every } j \text{ such that } i \le j < k, K_i^n(a) = \textsf{true}\\ \textsf{false} & \text{otherwise}\end{cases}
\end{aligned}$$

Now prove a totality lemma, which says that in any function K_i^n, every formula in $LTL_f(\mathbf{V})$ evaluates to **true** or **false**.

Lemma 3.3 (Totality)**.** *For every Kripke structure K^n, every $i \in \{1, \ldots, n\}$, and an arbitrary formula a, then $K_i^n(a) =$ **true** or $K_i^n(a) =$ **false**.*

Proof. By structural induction on an arbitrary formula a (Lemma A.1). $\triangle$

Example 3.4. As an example, we will define and motivate couple of common pieces of syntactic sugar [4, 21, 24]. Define

$$\Box\, a \triangleq a\, \mathcal{W}\, \bot \tag{3.1}$$

$$\Diamond\, a \triangleq \neg \Box \neg a \tag{3.2}$$

$$a\, \mathcal{U}\, b \triangleq a\, \mathcal{W}\, b \wedge \Diamond\, b \tag{3.3}$$

$$\bullet\, a \triangleq \neg \bigcirc \neg a \tag{3.4}$$

$$\mathsf{end} \triangleq \neg \bigcirc \top \tag{3.5}$$

Lets examine each of these new pieces of syntax in turn, examining why we might want to use them.

(3.1) The statement $\Box\, a$ can be interpreted as $K_i^n(\Box\, a) = \textbf{true}$ iff $K_j^n(\Box\, a) = \textbf{true}$ for all $i \leq j \leq n$. We encode this as $(a\ \mathcal{W}\ \bot)$. This eliminates the second case, since for any i, $K_i^n(\bot) = \textbf{false}$. Considering the first case, we have $K_j^n(a) = \textbf{true}$ for $i \leq j \leq n$, which is exactly when we want $K_i^n(\Box\, a)$ to be true.

(3.2) The unary operator $\Diamond\, a$, pronounced "ever a" or "eventually a" could be built into the fix K_i^n, interpreted as "there exists a state i such that $K_i^n(a) = \textbf{true}$". We can derive this algebraically.

$$\begin{aligned} K_i^n(\neg \Box \neg a) = \textbf{true} &\Leftrightarrow K_i^n(\Box \neg a) = \textbf{false} \\ &\Leftrightarrow \text{not for all } j, i \leq j \leq n \text{ such that } K_j^n(\neg a) = \textbf{true} \\ &\Leftrightarrow \text{exists } j, i \leq j \leq n \text{ such that } K_j^n(\neg a) = \textbf{false} \\ &\Leftrightarrow \text{exists } j, i \leq j \leq n \text{ such that } K_j^n(a) = \textbf{true} \end{aligned}$$

(3.3) The formula $a\, \mathcal{U}\, b$, pronounced "a until b" is equivalent to $a\, \mathcal{W}\, b$ except that we enforce that b is true at some point, hence the conjunctive $\Diamond\, b$.

(3.4) The next piece of syntax $\bullet\, a$, pronounced "weak next" is a form of next ($\bigcirc$) that has a valuation to **true** in the last state. We observe that $K_i^n(\bullet\, \top) = \textbf{true}$ forall states i, and this is not the case for $\bigcirc$. The operator behaves as expected for $i < n$. because of the double negation in the definition of $\bullet$. The interesting case coming when $i = n$, i.e. there is no next state. In the interpretation above, $K_n^n(\bigcirc\, \top) = \textbf{false}$, so $K^n_n(\bullet \top) = K^n_n(\neg \bigcirc \neg \top) = \textsf{true}$. In this way, the $\bullet$ operator is end-agnostic, and will still be true when the world has ended.

(3.5) This also justifies the $\textsf{end}$ sugar, which gives $\textsf{true}$ exactly when $i = n$. To see this lets evaluate $K^n_n(\textsf{end})$.

$$K^n_n(\textsf{end}) = K^n_n(\neg \bigcirc \top) = \textsf{true} \text{ iff } K^n_n(\bigcirc \top) = \textsf{false})$$

Since the only time $K^n_i(\bigcirc \top) = \textsf{false}$ is when $i = n$, we know that $K^n_n(\textsf{end}) = \textsf{true}$.

We have defined LTL$_f$'s evaluation scheme in a similar way to that of LTL, paying close attention to the end behavior. To further explore the nuances of LTL$_f$, we can compare the way that $\Box\, a$ behaves in LTL$_f$ to how it behaves in LTL. In LTL, the statement $\Box\, a$ means that in every single state now and in the future, a holds. Lets look at what happens when we want to examine $\Box\, a$ in the last state:

Lemma 3.5 (Always Convergence). *Given a Kripke structure of length n, and a formula a of LTL$_f$(**V**) then* $K^n_n(\Box\, a) = K^n_n(a)$

Proof. Immediate, by definition of the evaluation function (Lemma A.2). $\triangle$

This means that in the last state, given a formula a, if $K^n_n(a) = \textsf{true}$ then the formula a' created by removing all of the $\Box$ statements inside a gives $K^n_n(a') = \textsf{true}$. Then we only have to deal with $\bigcirc$-statements (which are false) and classical logic in the nth state.

With this denotational model of LTL over finite traces, we can define validity of a formula within the language $LTL_f(\mathbf{V})$. Notionally, this is when some assignment of truth values holds for every formula.

Definition 3.6 (Valid). A formula a of $LTL_f(\mathbf{V})$ is called *valid* in the temporal structure K^n for $\mathbf{V}$, denoted $\vDash_{K^n} a$, if $K^n_i(a) = \textsf{true}$ for every $i \in \mathbb{N}$.

Definition 3.7 (Consequence). For a set of formula $\textsf{false}$, we write $\mathcal{F} \vDash a$, "pronounced a is a consequence of $\textsf{false}$", if

$$\text{for all } K^n, (\text{for all } c \in \mathcal{F} \vDash_{K^n} b) \implies \vDash_{K^n} b$$

Definition 3.8 (Universal Validity). A formula a is called *universally valid* if a is a consequence the empty set of formulae. This is written $\emptyset \vDash a$, or equivalently $\vDash a$.

Remark 3.9. When we write K^n, we are not saying that n is an index to an infinite Kripke structure K or a finite one K^m with $m \geq n$. Rather, n is the number of time-steps in the structure, and we cannot formulate it without this n. It is helpful to think of n as an annotation that marks the structure's size.

Lets make a brief aside to consider an alternate notion of what it means to be a "finite Kripke structure". Instead of a finite list of valuations, we could define it as a finite prefix of an infinite kripke structure, parameterizing on n.

Definition 3.10 (Truncated Kripke Structure)**.** Let K be a kripke structure $K = (\eta_1, \eta_2, \ldots)$ where the functions η are defined as above, and let n be a positive integer. Then a *truncation* of K is a finite tuple $K[n] = [\eta_1, \eta_2, \ldots, \eta_n]$.

The definitions of Validity, Consequence, and Universal Validity would then mostly be syntactic translations from K^n to $K[n]$ assuming a Kripke structure K. However we need to modify consequence slightly:

Definition 3.11 (Consequence)**.** Let $\mathcal{F}$ be a set of formulae, a a formula, and K be a Kripke structure such that for every trucation of K at timestep $n \in \mathbb{N}$, and every formula $b \in \mathcal{F}$, we can write $\vDash_{K[n]} b$. Then write $\mathcal{F} \vDash a$, pronounced a is a consequence of **false**, if $\vDash_{K[n]} a$.

These two semantic models, finite Kripke structures and truncated Kripke structures, seem like they might have different consequences and different proof mechanisms. However, it doesn't matter which we choose, since they are, in fact, equivalent.

Corollary 3.12. *Given a finite Kripke structure $K^n = (\eta_1, \eta_2, \ldots, \eta_n)$ and an infinite Kripke structure $K = (\eta'_1, \eta'_2, \eta'_3, \ldots)$, such that $\eta_i = \eta'_i$, $\vDash_{K^n} a$ if and only if $\vDash_{K[n]} a$.*

Proof. By induction on the size of K^n (Lemma A.3). △

Now lets prove a couple of important lemmata that will help us later. First, we prove that implication behaves appropriately, by proving a semantic version of modus ponens.

Lemma 3.13 (Semantic Modus Ponens)**.** *For a given set of formulae $\mathcal{F}$, with a, b formulae, if $\mathcal{F} \vDash a$ and $\mathcal{F} \vDash a \rightarrow b$, then $\mathcal{F} \vDash b$.*

Proof. By unrolling the definitions (Lemma A.4). △

Now, we demonstrate the power of the temporal operators. We show that if you assume a, you will be able to show $\Box\, a$, and similarly, if you have a and $\bigcirc \top$, you can derive $\bigcirc\, a$.

Lemma 3.14 (Assumption of Temporal Operators)**.** *For a set of formulae $\mathcal{F}$, then $\mathcal{F} \cup \{a, \bigcirc \top\} \vDash \bigcirc\, a$ and $\mathcal{F} \cup \{a\} \vDash \Box\, a$.*

Proof. By definition; since $\mathcal{F} \vDash a$ means a is a consequence of $\mathcal{F}$, i.e. a holds at all steps (Lemma A.5). △

Corollary 3.15 (Always to Next)**.** *For a formula a, $\vDash (\bigcirc \top \wedge \Box\, a) \rightarrow \bigcirc\, a$.*

Proof. Direct consequence of the proof of Lemma 3.14. △

Corollary 3.16 (Next Assumption)**.**

$$\bigcirc \top \vDash \bot$$

Proof. Vacuously true — no finite Kripke structure exists such that $K_i^n(\bigcirc \top) =$ **true** for all $i \in \{1, \ldots, n\}$ (Lemma A.6). △

Notice that a consequence of Corollary 3.16 is that we can never assume $\bigcirc\, a$ for any a. We must bear this in mind when we formulate our axioms. In fact, it seems that when we assume a set of formulae $\mathcal{F}$ we are lifting the $\Box$ operator over the set and assuming the formulae in the set $\{\Box\, a \mid a \in \mathcal{F}\}$. Notice that this means that our idea of deduction for the $\vDash$ relation must be slightly different than we might usually imagine. Typically, we are able to show that $a \vDash b$ if and only if $\vDash a \rightarrow b$. However, Lemma 3.14 demonstrates that we won't get this result. In fact, the best we can do is $a \vDash b$ if and only if $\vDash (\Box\, a) \rightarrow b$ as shown in Theorem 3.17.

Theorem 3.17 (Semantic Deduction)**.** *For all $\mathcal{F}$, a and b,*

$$\mathcal{F} \cup \{a\} \vDash b, \text{ if and only if } \mathcal{F} \vDash (\Box\, a) \rightarrow b.$$

Proof. First prove $\mathcal{F} \cup \{a\} \vDash b \implies \mathcal{F} \vDash \Box\, a \rightarrow b$ and then prove $\mathcal{F} \vDash \Box\, a \rightarrow b \implies \mathcal{F} \cup \{a\} \vDash b$.

($\Rightarrow$) For the forwards direction, assume $\mathcal{F} \cup \{a\} \vDash b$, and let

$$K^n = (\eta_1, \eta_2, \ldots, \eta_n)$$

be a temporal structure such that $\vDash_{K^n} c$ for every $c \in \mathcal{F}$. Let $i \in \{1, \ldots, n\}$. Then $K_i(c) = \textsf{true}$ for every $c \in \mathcal{F}$. We want to show that $K_i^n(\Box\, a \to b)$. If $K_i^n(\Box\, a) = \textsf{false}$ the result is trivial, so assume that $K_i^n(\Box\, a) = \textsf{true}$ which means that $K_j^n(a) = \textsf{true}$ for every $j \geq i$, to show that $K_i^n(b) = \textsf{true}$.

Define $(K^n)^{(i)} = (\eta'_1, \eta'_2, \cdots \eta'_{n-i})$ to be the temporal structure constructed from K^n such that $\eta'_j = \eta_{i+j}$.

Now we get $(K^n)^{(i)}_j(a) = \textsf{true}$ and $(K^n)^{(i)}_j(c)$ for all $0 \leq j \leq n - i$ and $c \in \mathcal{F}$. Then by assumption we get $(K^n)^{(i)}_0(b) = \textsf{true}$, which is equivalent to $K_i^n(b) = \textsf{true}$.

($\Leftarrow$) Assume that $\mathcal{F} \vDash \Box\, a \to b$. Define $K^n = (\eta_1, \cdots, \eta_n)$ to be a finite temporal structure such that $\vDash_{K^n} c$ for every $c \in \mathcal{F} \cup \{a\}$ and $i \in [n]$. Then, $K_i^n(\Box\, a \to b) = \textsf{true}$ and $K_j^n(a) = \textsf{true}$ for every $j \geq i$. So by definition $K_i^n(\Box\, a)$. Then by Lemma 3.13, $K_i^n(b) = \textsf{true}$ meaning that $\vDash_{K^n} b$ so $\mathcal{F} \cup \{a\} \vDash b$.

$\triangle$

Finally, we will demonstrate a common sense Unsatisfiability Lemma (Lemma 3.18), which says that for a valid statement a, there is no finite Kripke structure in which a evaluates to **true**.

Lemma 3.18 (Unsatisfiability)**.** *If* $\vDash a$*, then there for every* K^n *and every* $i \in \{1, \ldots, n\}$*,* $K_i^n(\neg a) = \textsf{false}$*. We say that* $\neg a$ *is* unsatisfiable*.*

Proof. By the definition of the universal validity, for every Kripke structure $K_i^n(a) = \textsf{true}$, for every $i = 1, \ldots, n$, so by definition of the Kripke valuation function, $K_i^n(\neg a) = \textsf{false}$. So, there is no Kripke structure that satisfies $\neg a$. $\triangle$

3.3 Axiomatization of LTL$_f$

Now, we must define a formal system for LTL$_f$. We will assume all tautologies from propositional logic as axioms. We will refer to this axiom as Taut. Then we will define several axioms modified from LTL.

Definition 3.19 (Formal System for LTL$_f$)**.** For a set of variables $\mathbf{V}$, define the relation $\cdot \vdash \cdot \subseteq 2^{LTL_f(\mathbf{V})} \times LTL_f(\mathbf{V})$ by the rules in Figure 3.2. Instead of writing $\emptyset \vdash a$, we write $\vdash a$. Sometimes, instead of writing $\vdash a$ for a formula a, we say *a holds.* Also, for a formula $\mathcal{F}$, we write $\mathcal{F} \vdash a$, if, by assuming $\vdash c$ for all $c \in \mathcal{F}$, we can derive $\vdash a$.

$$\text{all propositional tautologies} \qquad (\textsc{Taut})$$

$$\vdash \bullet(a \to b) \leftrightarrow (\bullet a \to \bullet b) \qquad (\textsc{WkNextDistr})$$

$$\vdash \mathsf{end} \to \neg \bigcirc a \qquad (\textsc{EndNextContra})$$

$$\vdash \Diamond \mathsf{end} \qquad (\textsc{Finite})$$

$$\vdash a \mathcal{W} b \leftrightarrow b \lor (a \land \bullet(a \mathcal{W} b)) \qquad (\textsc{WkUntilUnroll})$$

$$\frac{\vdash a}{\vdash \bullet a} \qquad (\textsc{WkNextStep})$$

$$\frac{\vdash a \to b \quad \vdash a \to \bullet a}{\vdash a \to \Box b} \qquad (\textsc{Induction})$$

Figure 3.2: Axioms & Inference Rules for LTL$_f$

Figure 3.3 presents some useful rules derived from the axioms and inference rules [4, 21, 25]. Their full proofs are presented in Appendix A. The judgments presented in Figure 3.3 are the kinds of formulae we can derive in the proof system. However, we want to make sure that all of these properties, and indeed the axioms and inference rules themselves, make statements about the model-theoretic judgement, i.e. we want to show the axioms above are *sound.*

Theorem 3.20 (Soundness Theorem for LTL$_f$)**.** *Let a be a formula. If $\vdash a$, then $\vDash a$.*

Proof. By structural induction on $\vdash a$, only showing the cases that explicitly deal with the end of time:

(EndNextContra) We want to show that $\vDash \neg \bigcirc \top \to \neg \bigcirc a$. So given some finite Kripke structure K^n, we have two cases, either $K^n_i(\mathsf{end}) =$

$\vdash \bigcirc a \rightarrow \neg\mathsf{end}$	NextNotEnd
$\vdash \neg(\bigcirc \top \wedge \bigcirc \bot)$	NextContraIsContra
$\vdash \neg \bigcirc a \leftrightarrow \mathsf{end} \vee \bigcirc \neg a$	CommNegNext
$\vdash \bullet\, a \leftrightarrow (\bigcirc a \vee \mathsf{end})$	NextWkNext
$\vdash (\mathsf{end} \vee \bigcirc(a \rightarrow b)) \leftrightarrow (\bigcirc a \rightarrow \bigcirc b)$	NextDistr
$\vdash \bullet\, \top$	WkNextTop
$\dfrac{\vdash a \rightarrow b}{\vdash \bigcirc a \rightarrow \bigcirc b}$	NextMonotone
$\vdash \Box\, a \leftrightarrow a \wedge \bullet\, \Box\, a$	AlwUnroll
$\vdash \Diamond\, a \leftrightarrow a \wedge \bigcirc \Diamond\, a$	EverUnroll
$\vdash (\Box\, a \wedge \Box\, b) \rightarrow \Box(a \wedge b)$	AlwaysAndDistr
$\dfrac{\vdash a \rightarrow b}{\vdash \Box\, a \rightarrow \Box\, b}$	AlwMonotone
$\vdash \Box\, a \wedge \Diamond\, b \rightarrow \Diamond(b \wedge a)$	AlwaysEver
$\vdash \Box\, a \rightarrow \Diamond(\mathsf{end} \wedge a)$	AlwaysFinite
$\vdash (a\, \mathcal{U}\, b) \leftrightarrow b \vee (a \wedge \bigcirc(a\, \mathcal{W}\, b)))$	UntilUnroll
$\vdash \neg(a\, \mathcal{U}\, b) \leftrightarrow (\neg b)\ \mathcal{W}\ (\neg \wedge \neg b)$	NotUntil
$\vdash \Box\, a \leftrightarrow \neg \Diamond \neg a$	AlwaysEverDual
$\dfrac{\vdash c \rightarrow b \vee (a \wedge \bullet\, c)}{\vdash c \rightarrow (a\, \mathcal{W}\, b)}$	WkUntilInduction
$\vdash \Box\, a \rightarrow a\, \mathcal{W}\, b$	AlwaysWkUntil

Figure 3.3: Derived rules (proofs in Appendix A)

false and we're done, or $K_i^n(\neg \bigcirc \top) =$ **true** and we need to show that $K_i^n(\neg \bigcirc a) =$ **true**. The only way that $K_i^n(\neg \bigcirc \top)$ could be **false** is if $i = n$, so $K_i^n(\bigcirc a) = K_n^n(\bigcirc a) =$ **false**. Then applying the definition gives $K_i^n(\neg \bigcirc a) =$ **true**.

(Finite) Assume we have a proof that $\vdash \Diamond \mathsf{end}$. We want to show that $\vDash \Diamond \mathsf{end}$. We can desugar this to a form that the K_i^n functions will understand, namely $\vDash \neg \Box \bigcirc \top$. Take an arbitrary function K_i^n and show that $K_i^n(\neg((\bigcirc \top)\ \mathcal{W}\ \bot)) =$ **true**. Equivalently, we show that $K_i^n(((\bigcirc \top)\ \mathcal{W}\ \bot)) =$ **false**. Since we know that $K_k^n(\bot) =$ **false** for all possible k, we must find an index $j \in [i, n]$ such that $K_j^n(\bigcirc \top) =$ **false**. Let $j = n$, then by definition $K_j^n(\bigcirc \top) = K_n^n(\bigcirc \top) =$ **false**.

See Lemma A.7 for remaining cases.

△

Theorem 3.17 showed that for a set of formulae $\mathcal{F}$, and formulae a, b we have $\mathcal{F} \cup \{a\} \vDash b$ if and only if $\mathcal{F} \vDash (\Box a) \to b$. We want to be able to show the exact same thing, swapping the $\vdash$ relation for the $\vDash$ relation.

This informs how we can perform deductive proofs in the logic. In Classical logic, $\{a\} \vdash b$ if and only if $\vdash a \to b$. So, we prove statements of the form $\vdash a \to b$ by first proving the equivalent $\{a\} \vdash b$; we say "Assume $\vdash a$, and show $\vdash b$". However in our proof system for LTL$_f$, the judgement $a \vdash b$ means $\vdash (\Box a) \to b$ which does not prove $\vdash a \to b$. Hence, we cannot derive $\vdash a \to b$ by saying "Assume $\vdash a$ and show $\vdash b$". See the proofs in Appendix A for examples of how to resolve this trouble.

Theorem 3.21 (Deduction Theorem)**.** *Let a, b be formulae, $\mathcal{F}$ a set of formulae. $\mathcal{F} \cup \{a\} \vdash b$ if and only if $\mathcal{F} \vdash \Box a \to b$.*

Proof. From left to right, by induction on the derivation of $\mathcal{F} \cup \{a\} \vdash b$; from right to left, by Taut, WkNext, and Induction (c.f. Theorem A.8). △

3.4 Completeness for LTL$_f$

We want to show completeness of LTL$_f$, i.e. we want to show that $\forall K^n, \vDash_{K^n} a$ implies that $\vdash a$, for all terms a. That is, if something is valid in the model, we can prove it using our axioms.

Our proof largely follows the completeness proof for LTL [21], except we make sure to both ensure the existence of a final state, and in constructing satisfying timelines, we make sure to keep track of whether we are in a final state.

To show the existence of a proof for a formula a, we create a PNP based on the negation of this formula. As in the proof of Theorem 2.13, this PNP should be $P = (\emptyset, \{a\})$. We will want to show that P is inconsistent, i.e. that we can derive $\vdash \neg\neg a$, which is exactly $\vdash a$. However, we need to ensure that our construction respects the finiteness assumption, so we will inject the axiom Finite into the positive set of this PNP. Construct another PNP $\mathcal{P} = (\{\Diamond\, \mathsf{end}\}, \{a\})$; and demonstrate it's inconsistency. To do this, we use a satisfiability theorem (Theorem 3.42) that says that consistent PNPs have satisfiable interpretations. We show $\mathcal{P}$ is unsatisfiable, to conclude that it is inconsistent. This means that $\vdash \neg\mathcal{P}$, so $\vdash \neg(\Diamond \neg \bigcirc \top \wedge \neg a)$, which demonstrates $\vdash a$ using Taut.

The key point in this proof is the satisfiability theorem (Theorem 2.13) which says that every consistent PNP is satisfiable. To show this theorem we will construct a large graph structure from a PNP $\mathcal{P}$, called $\mathcal{G}_{\mathcal{P}}$, that will step the PNP through time, examining all potential successors at each time step. Each node of the graph is a consistent PNP, we will use a closure function called τ and the step function called σ to create the edge set in the graph.

We inductively define the function $\tau : LTL_f(\mathbf{V}) \to \mathbf{2}^{LTL_f(\mathbf{V})}$ from a formula to a set of formulae, to be set of all subformulae of a given formula (taking $\bigcirc a$ as atomic). We can then lift this to sets of formulae and PNPs in the obvious way. For a given PNP $\mathcal{Q}$, call $\tau(\mathcal{Q})$ the closure of $\mathcal{Q}$. For example, we can consider a PNP $\mathcal{Q}_0 = (\{\Diamond\, \mathsf{end}, \bigcirc \top\}, \emptyset)$, then $\tau(\mathcal{Q}_0)$ will contain every recursive expansions of the formulae in $\mathcal{Q}_0$, we see this below:

$$\tau(\mathcal{Q}_0) = \{\Diamond\, \mathsf{end}, \Box\, \neg\mathsf{end}, \neg\mathsf{end}, \neg \bigcirc \top, \bigcirc \top\}$$

So, for a given PNP $\mathcal{Q}$, we want to take all of the formula in $\tau(\mathcal{Q})$ and construct all sensible assignments of the subformulae in $\tau(\mathcal{Q})$ to the positive or negative sets of a PNP. Specifically, extend the positive and negative sets of $\mathcal{Q}$ with elements of $\tau(\mathcal{Q}) - \mathcal{F}_{\mathcal{Q}}$, where no formula is in both sets, to get a to get a new PNP $\mathcal{Q}^*$. We say that $\mathcal{Q}^*$ is a completion of $\mathcal{Q}$. So, continuing the above example, one possible completion of $\mathcal{Q}_0$ is $\mathcal{Q}_0^* = (\{\neg\mathsf{end}, \Diamond\, \mathsf{end}, \bigcirc \top\}, \{\bot, \mathsf{end}, \Box\, \neg\mathsf{end}\})$. We see that $pos(\mathcal{Q}_0) \subset pos(\mathcal{Q}_0^*)$ and

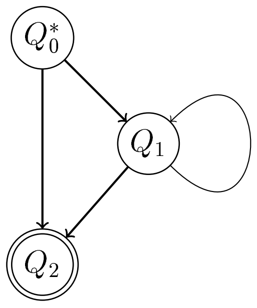


Figure 3.4: An Example Proof Graph

since $neg(\mathcal{Q}_0) = \emptyset$, the same is true for the negative sets. Notice also that the two sets are distinct, and every element of $\tau(\mathcal{Q}_0)$ appears exactly once.

To get the successors of a PNP, we will step the PNP forward one step in time, and then take all of the completions at that step. This step function is called σ. So now, we can to step this forward one place to get

$$\sigma(\mathcal{Q}_0^*) = (\{\top\}, \{\Box \neg \mathsf{end}\})$$

Then we can find all consistent completions of $\sigma(\mathcal{Q}_0^*)$, these will be the successors of $\mathcal{Q}_0^*$. They are:

$$\mathcal{Q}_1 = (\{\top, \neg\mathsf{end}, \bigcirc \top\}, \{\bot, \mathsf{end}, \Box \neg\mathsf{end}\})$$
$$\mathcal{Q}_2 = (\{\top, \mathsf{end}\}, \{\bot, \neg\mathsf{end}, \bigcirc \top, \Box \neg\mathsf{end}\})$$

These two PNPs, $\mathcal{Q}_1$ and $\mathcal{Q}_2$ correspond to the interesting successor states, where time ends in $\mathcal{Q}_1$ but continues in $\mathcal{Q}_2$.

If we continue to build this tree by finding the consistent completions of $\sigma(\mathcal{Q}_1)$ and $\sigma(\mathcal{Q}_2)$, we see that $\sigma(\mathcal{Q}_2)$ has no consistent completions, whereas the consistent completions of $\sigma(\mathcal{Q}_1)$ are exactly the same as the consistent completions of $\sigma(\mathcal{Q}_0)$. Then we can construct the graph, $\mathcal{G}_{\mathcal{Q}_0^*}$ shown in Figure 3.4. We can see the satisfying models in $\mathcal{G}_{\mathcal{Q}_0^*}$, either the next state is the end of time $\mathcal{Q}_0$ to $\mathcal{Q}_2$, or you can take arbitrarily many steps, using any path like $\pi = \mathcal{Q}_0, \mathcal{Q}_1, \ldots, \mathcal{Q}_1, \mathcal{Q}_2$. This path corresponds to a finite Kripke structure $K^{|\pi|} = (\eta_1, \ldots, \eta_{|\pi|}$ where $\eta_i(v) = $ **true** if and only if v is in the positive set of the ith node in π.

Notice in particular that $\mathcal{G}_{\mathcal{Q}_0*}$ is finite. In general, the set of nodes in an arbitrary proof graph is finite, and for our original PNP $\mathcal{P} = (\{\Diamond \mathsf{end}\}, \{a\})$, we engage in a path-finding exercise[1]. We care about *terminal* paths. A terminal path is one that ends in a node $\mathcal{Z}$, such that $\bigcirc\top \in \mathcal{Z}$ (called a *terminal node*). In our running example, $\mathcal{Q}_2$ is a terminal node, and is identified with a double circle in $\mathcal{G}_{\mathcal{Q}_0^*}$. So, all paths that end in $\mathcal{Q}_2$ are terminal, i.e. all paths of the form $\mathcal{Q}_0, \underbrace{\mathcal{Q}_1, \ldots \mathcal{Q}_1}_{\text{0 or more}}, \mathcal{Q}_2$.

Some proofs for completeness of LTL [21, 24] try to find something called a *fulfilling* or *complete* path, by which they mean that all temporal formulae in the path are realized in the path. In the finite setting, it is straightforward to prove that terminal paths are exactly the fulfilling paths (Corollary 3.40).

Now, we know there exists a terminal path $\pi = P_1, P_2, \ldots, P_n$. We create a Kripke structure K^n from this path as above. We then show that $K_1^n(a) =$ **true**, i.e. that a is satisfiable. So we can conclude that for every PNP $\mathcal{Q}$, if $\mathcal{Q}$ is consistent, then $\widehat{\mathcal{Q}}$ is satisfiable.

So how do we use the satisfiability result to prove the general completeness result? We know that if a PNP is consistent, then it is satisfiable. We've created this PNP $\mathcal{P} = (\{\Diamond \mathsf{end}\}, \{a\})$, with the intent to show that $\models a$ implies $\vdash a$. We know that $\neg a$ is unsatisfiable since a is valid, so we know that $\mathcal{P}$ is inconsistent. This gives us a derivation of $\vdash \neg(\Diamond \mathsf{end} \wedge \neg a)$, which by some classical machinery gives $\vdash a$.

Now we proceed with this proof formally.

Preliminary Definitions and Lemmata

Before we can *use* the PNPs we defined in Definition 2.11, we need to prove a lemma about the way that they behave.

Lemma 3.22 (PNPs are Well-Behaved)**.** *Let $\mathcal{P} = (\mathcal{F}^+, \mathcal{F}^-)$ be a consistent PNP, and a, b be formulae, then*

1. *$\mathcal{F}^+$ and $\mathcal{F}^-$ are disjoint*
2. *Either $(\mathcal{F}^+, \mathcal{F}^- \cup \{a\})$ or $(\mathcal{F}^+ \cup \{a\}, \mathcal{F}^-)$ is a consistent PNP*
3. *$\bot \notin \mathcal{F}^+$*
4. *If, $a, b, a \to b \in \mathcal{F}_{\mathcal{P}}$, then if $a \in \mathcal{F}^-$ or $b \in \mathcal{F}^+$, $a \to b \in \mathcal{F}^+$, otherwise $a \to b \in \mathcal{F}^-$.*

[1]Here we are using a loose definition of *path* where repetition and self-loops are allowed [21]. Some call this structure a *walk* [31]

5. *If $a \to b, a, b \in \mathcal{F}_{\mathcal{P}}$, then if $a \to b \in \mathcal{F}^+$, and $a \in \mathcal{F}^+$, then $b \in \mathcal{F}^+$.*

Proof. See Lemma A.10. △

We also want a way to reason about how a formula evolves over time. We will create a graph structure whose nodes are consistent PNPs, and the edges represent a potential step forward in time. To do this, we will define a "step" function σ that will mirror the way the K^n function steps through the lifetime of the temporal operators.

Definition 3.23 (Step Function)**.** The *step function*, σ is inductively defined for a PNP $P = (\mathcal{F}^+, \mathcal{F}^-)$ as follows:

$$\begin{aligned}
\sigma_1^+(\mathcal{P}) &= \{a \mid \bigcirc a \in pos(\mathcal{P})\} \\
\sigma_2^+(\mathcal{P}) &= \{a \,\mathcal{W}\, b \mid a \,\mathcal{W}\, b \in pos(\mathcal{P}) \text{ and } b \in neg(\mathcal{P})\} \\
\sigma_3^+(\mathcal{P}) &= \{a \,\mathcal{W}\, b \mid a \,\mathcal{W}\, b \in pos(\mathcal{P}), \bigcirc \top \in neg(\mathcal{P}), \text{ and } a \in pos(\mathcal{P})\} \\
\sigma_4^-(\mathcal{P}) &= \{a \mid \bigcirc a \in neg(\mathcal{P})\} \\
\sigma_5^-(\mathcal{P}) &= \{a \,\mathcal{W}\, b \mid a \,\mathcal{W}\, b \in neg(\mathcal{P}) \text{ and } a \in pos(\mathcal{P}) \text{ or } b \in pos(\mathcal{P})\}
\end{aligned}$$

Where $\sigma(\mathcal{P}) = (\sigma_1^+(\mathcal{P}) \cup \sigma_2^+(\mathcal{P}) \cup \sigma_3^+, \sigma_4^-(\mathcal{P}) \cup \sigma_5^-(\mathcal{P}))$.

We call a formula $f \in \mathcal{F}_{\mathcal{P}}$ *unresolved* in a PNP $\mathcal{P}$ if $f \in \mathcal{F}_{\sigma(\mathcal{P})}$; otherwise, call f *resolved*.

Intuitively, applying σ to a function steps all formulae of the form $\bigcirc a$ to a, in their respective sets, and then for each statement like $a \,\mathcal{W}\, b$, it assesses whether we have seen as at every step until we see our first b. These actions combine to provide a transformation on PNPs that morally represents taking one step forwards in time.

Example 3.24. Now, we can formally see the way that the σ function works for the node $\mathcal{Q}_0^*$ that we examined earlier. Recall the PNP:

$$\mathcal{Q}_0^* = (\{\neg \mathsf{end}, \Diamond \mathsf{end}, \bigcirc \top\}, \{\bot, \mathsf{end}, \Box \neg \mathsf{end}\}) .$$

Since our definitions are written in terms of the syntactic sugar, lets reduce the formulae in $\mathcal{Q}_0^*$ to only use the basic operators. Then, we rewrite $\mathcal{Q}_0^*$ as

$$\mathcal{Q}_0^* = (\{\neg\neg \bigcirc \top, \neg((\neg\neg \bigcirc \top) \,\mathcal{W}\, \bot), \bigcirc \top\}, \{\bot, \neg \bigcirc \top, (\neg\neg \bigcirc \top) \,\mathcal{W}\, \bot\}) .$$

We compute:

$$\sigma(\mathcal{Q}_0^*) = (\{\top\}, \{(\neg\neg \bigcirc \top)\ \mathcal{W}\ \bot\}) \equiv (\{\top\}, \{\Box \neg \mathsf{end}\}).$$

Notice that $\bigcirc \top$ is *resolved*, and $(\neg\neg \bigcirc \top)\ \mathcal{W}\ \bot$ is *unresolved.*

We want this σ-function to have a couple of essential properties, and the 4-part definition facilitates a proof by cases. First of all, the σ function is intended to simulate the change in interesting formulae over a single time step. In order for this representation to make sense, $\vdash \widehat{\mathcal{P}}$ must imply $\vdash \bigcirc \widehat{\sigma(\mathcal{P}))}$ — unless we're at the end of time, in which case we get $\neg \bigcirc a$ for all formulae a. So, to encompass these two cases, we show that $\widehat{\mathcal{P}} \to \bullet \widehat{\sigma(P)}$.

Lemma 3.25 (Next-Step Implication)**.** *Let $\mathcal{P}$ be a PNP*

$$\vdash \widehat{\mathcal{P}} \to \bullet \widehat{\sigma(\mathcal{P})}$$

Proof. By cases on σ_i, $\vdash \widehat{\mathcal{P}} \to \bullet c$ whe have $c \in \sigma_1(\mathcal{P}) \cup \sigma_2(\mathcal{P})$, and $\vdash \widehat{\mathcal{P}} \to \bullet \neg c$ when $c \in \sigma_3(\mathcal{P}) \cup \sigma_4(\mathcal{P})$. The proposition follows directly from this and WkNextAndDistr. See Lemma A.9 for the case analysis.

△

Lemma 3.26 (Step Consistency)**.** *Let $\mathcal{P}$ be a consistent PNP. If $\vdash \widehat{\mathcal{P}} \to \neg\mathsf{end}$, then $\sigma(\mathcal{P})$ is consistent.*

Proof. First we show that $\vdash \widehat{\mathcal{P}}$ implies $\vdash \widehat{\sigma(\mathcal{P})}$. So, assume $\vdash \widehat{\mathcal{P}}$, and for the sake of contradiction, assume $\vdash \neg\widehat{\sigma(\mathcal{P})}$. Then, from WkNext we get $\vdash \bullet \neg\widehat{\sigma(\mathcal{P})}$. Then, finally from NextWkNext we get $\vdash \mathsf{end} \vee \neg \bullet \widehat{\sigma(\mathcal{P})}$. However, we know that $\vdash \widehat{\mathcal{P}} \to \neg\mathsf{end}$, so we can just conclude $\vdash \bullet \widehat{\sigma(\mathcal{P})}$. Then by the contrapositive of Lemma 3.25, derive $\vdash \neg\widehat{\mathcal{P}}$. We have a contradiction with $\vdash \widehat{\mathcal{P}}$, so conclude that $\vdash \widehat{\sigma(\mathcal{P})}$.

So, if $\mathcal{P}$ is consistent, then we know that it is not the case that $\vdash \widehat{\mathcal{P}}$. We want to show that $\sigma(\mathcal{P})$ is consistent, so assume, for the sake of contradiction, that $\vdash \neg\widehat{\sigma(\mathcal{P})}$. Then by the contrapositive of $\vdash \widehat{\mathcal{P}}$ implies $\vdash \widehat{\sigma(P)}$ (shown in the previous paragraph), we get $\vdash \neg\widehat{\mathcal{P}}$, which contradicts the assumption of $\mathcal{P}$'s consistency. Hence $\sigma(\mathcal{P})$ is consistent. △

Now that we have a way of stepping PNPs through timesteps, we want to be able to examine all the pieces of a PNP to see how they interact with each other. The tool we need to do this is the *completion* of a PNP. It allows us to break apart composite formulae into their atomic pieces and assign each subformula to the positive or negative set. This gives *all* of the information about how a PNP is consistent. We define the function τ to perform this examination of subformulae, formalize a notion of *completion*, and define the comps function.

Definition 3.27 (Closure)**.** Define the *closure function* $\tau : \mathcal{F} \to 2^{\mathcal{F}}$ as a fixpoint on an input formula, that returns the set of subformula of the input:

$$\begin{aligned}
\tau(v) &= v && \text{for } v \in \mathbf{V} \\
\tau(\bot) &= \{\bot\} \\
\tau(a \to b) &= \{a \to b\} \cup \tau(a) \cup \tau(b) \\
\tau(\bigcirc\, a) &= \{\bigcirc\, a\} \\
\tau(a \mathcal{W} b) &= \{a \mathcal{W} b\} \cup \tau(a) \cup \tau(b)
\end{aligned}$$

We can also lift the function τ over sets of formula, i.e. $\tau : 2^{\mathcal{F}} \to 2^{\mathcal{F}}$, using the definition $\tau(\mathcal{F}) = \bigcup_{f \in \mathcal{F}} \tau(f)$, and to apply over a PNP, where $\tau(\mathcal{P}) = \tau(\mathcal{F}_{\mathcal{P}})$. The set $\tau(x)$ is called the *closure of x*.

Definition 3.28 (Extensions and Completions)**.** A PNP $\mathcal{Q}$ is an *extension* of another PNP $\mathcal{P}$ if $pos(\mathcal{P}) \subseteq pos(\mathcal{Q})$ and $neg(\mathcal{P}) \subseteq neg(\mathcal{Q})$. We say $\mathcal{Q}$ *extends* $\mathcal{P}$. $\mathcal{Q}$ is a *completion* of $\mathcal{P}$ if $\tau(\mathcal{P}) = pos(\mathcal{Q}) \cup neg(\mathcal{Q})$ and $\mathcal{Q}$ is an extension of $\mathcal{P}$; $\mathcal{Q}$ is called *complete*.

Definition 3.29 (Completion Function)**.** Let $\mathcal{F} \in LTL_f(\mathbf{V})$ be a set of formulae and let the function $\mathsf{comps} : 2^{LTL_f(\mathbf{V})} \to 2^{PNP}$ be defined as

$$\mathsf{comps}(\mathcal{F}) = \{\mathcal{Q} \mid \tau(\mathcal{Q}) = \tau(\mathcal{F})\}$$

We overload the definition of comps to also be a function $\mathsf{comps} : PNP \to 2^{PNP}$ such that:

$$\mathsf{comps}(\mathcal{P}) = \{\mathcal{Q} \mid \tau(\mathcal{Q}) = \tau(\mathcal{P}) \text{ and } \mathcal{Q} \text{ extends } \mathcal{P}\}$$

The completion of a PNP is an expansion of all of its subformulae that can be evaluated at the current time step, i.e. all of those formulae that are not under a $\bigcirc$ operator. We want to be able to say something about the consistency of the completions of a complete PNP. But which completions are consistent?

Example 3.30. Given a PNP $\mathcal{P} = (\{v_1 \to v_2\}, \emptyset)$, where v_1 and v_2 are variables, we have four completions:

$$\begin{aligned}
\mathsf{comps}(\mathcal{P}) &= \{\mathcal{P}_0^*, \mathcal{P}_1^*, \mathcal{P}_2^*, \mathcal{P}_3^*\}\\
\mathcal{P}_0^* &= (\{v_1 \to v_2, v_1, v_2\}, \emptyset)\\
\mathcal{P}_1^* &= (\{v_1 \to v_2, v_1\}, \{v_2\})\\
\mathcal{P}_2^* &= (\{v_1 \to v_2, v_2\}, \{v_1\})\\
\mathcal{P}_3^* &= (\{v_1 \to v_2\}, \{v_1, v_2\})
\end{aligned}$$

All of these are consistent, except for $\mathcal{P}_2^*$, the axiom Taut tells us that the formula $\widehat{\mathcal{P}_2^*} \equiv v_1 \to v_2 \wedge v_2 \wedge \neg v_1$ is contradictory.

The previous example demonstrates that *not all* completions are consistent, but we'd really like it if *at least one* of the completions were consistent (Lemmata 3.31 and 3.33).

Lemma 3.31 (Consistent Closure Existence)**.** *Let $\mathcal{F}$ be a finite set of formulae, and let $\mathcal{P}_1^*, \cdots \mathcal{P}_m^*$ be PNPs with $\mathcal{F}_{\mathcal{P}_i^*} = \tau(\mathcal{F})$ for all $i \in \{1, \dots, m\}$ such that the positive and negative sets are disjoint. Then, $\vdash \bigvee_{i=1}^m \widehat{\mathcal{P}_i^*}$.*

Proof. By induction on the number of formulae in $\mathcal{F}$. Consider the base case where $\mathcal{F} = \tau(\mathcal{F}) = \emptyset$. Then $\mathcal{P}^* = (\emptyset, \emptyset)$ so with $\mathcal{F}_{\mathcal{P}^*} = \tau(\mathcal{F})$, and $\widehat{\mathcal{P}^*} \equiv \top$ and the conclusion holds by Taut.

Now consider the case where $\tau(\mathcal{F}) = \{a_1, \dots, a_k\}$, for $k \geq 1$. Since $\tau(f)$ only contains f and subterms of f, there is some maximal formula a_j such that

$$a_j \notin \tau(\{a_1, \dots, a_{j-1}, a_{j+1}, \dots, a_k\}).$$

Let $\mathcal{F}' \equiv \tau(\mathcal{F}) - a_j$. Let ${\mathcal{P}^*}'_1, \dots, {\mathcal{P}^*}'_l$, be the PNPs constructed from $\mathcal{F}'$. Then for each PNP ${\mathcal{P}^*}'_i$ we construct two new PNPs, one by adding a_j to the positive set, and one by adding a_j to the negative set. Let $m = 2l$ and the PNPs $\mathcal{P}_1^*, \dots, \mathcal{P}_m^*$ be those PNPs constructed in this manner. The induction hypothesis is that $\vdash \bigvee_{i=1}^l {\mathcal{P}^*}'_i$. So using the construction above, and Lemma 3.22, we can conclude that $\vdash \bigvee_{i=1}^l ({\mathcal{P}^*}' \vee a_j) \vee \bigvee_{i=1}^l ({\mathcal{P}^*}' \vee \neg a_j)$. The result follows by Taut. △

Example 3.32. Let $\mathcal{F} = \{a \wedge \neg a\}$. Show that we can prove one of the PNPs constructed from its completeions. We can do this despite the fact that $a \wedge \neg a$ is neither provable nor satisfiable. Calculate $\tau(\mathcal{F})$ to be

$$\tau(\mathcal{F}) = \{a \land \neg a, a, \neg a\}$$

Let $\mathcal{P}_i^*$ for $i = 1, \ldots, m$ be the PNPs that can be constructed by partitioning $\tau(\mathcal{F})$. To show $\vdash \bigvee_{i=1}^m \widehat{\mathcal{P}_i^*}$, we only need to show $\vdash \widehat{\mathcal{P}_i^*}$ for some $i = 1, \ldots m$. We know there is some i such that $\mathcal{P}_i^* = (\{\neg a\}, \{a, (a \land \neg a)\}$. Note that $\widehat{\mathcal{P}_i^*} = \neg a \land \neg a \land \neg(a \land \neg a)$. So, by TAUT, we can see that $\vdash \widehat{\mathcal{P}_i^*}$. conclude $\vdash \bigvee_{i=1}^m \widehat{\mathcal{P}_i^*}$.

Lemma 3.33 (Consistent Completion Existence)**.** *For $\mathcal{P}$, a consistent PNP, and*

$$\vdash \widehat{\mathcal{P}} \to \bigvee_{\mathcal{Q} \in \mathsf{comps}(\mathcal{P})} \widehat{\mathcal{Q}}$$

Proof. Let $\mathcal{P}_1, \ldots, \mathcal{P}_m$ be those PNPs with disjoint *pos*/*neg* sets such that $\tau(\mathcal{P}) = \tau(\mathcal{P}_i)$. Each one is either in $\mathsf{comps}(\mathcal{P})$, inconsistent, or not an extension, for more detail see Lemma A.11. △

Lemmata 3.31 and 3.33 allow us to construct the graph using σ and comps to create the edge relation. Because now we know that for all PNPs that dont represent the end of time, we can find a consistent successor (Lemmata 3.33 and 3.26) that represents a next state (Lemma 3.25).

Definition 3.34. For a consistent and complete PNP $\mathcal{P}^*$, define the *proof graph rooted at* $\mathcal{P}^*$, called $\mathcal{G}_{\mathcal{P}^*}$, to be the root $\mathcal{P}^*$, and the successors $\mathcal{G}_{\mathcal{Q}}$ for every consistent $\mathcal{Q} \in \mathsf{comps}(\mathcal{P}^*)$.

Corollary 3.35. *For a consistent and complete PNP $\mathcal{P}^*$, there are a finite number of vertices in $\mathcal{G}_{\mathcal{P}^*}$.*

Proof. Each node is a PNP. Specifically each PNP has a finite number of children, because there are a finite number of completions via τ, and there are a finite number of unique steps as the σ function consumes $\bigcirc$ operators, negative $\mathcal{W}$ operators, and continuing positive $\mathcal{W}$ operators. Since our inital formula is constrained to be finite, we can only have finitely many nodes. △

Intuitively this graph $\mathcal{G}_{\mathcal{P}}$ represents the potential satisfying Kripke structures of the original formulae in the root $\mathcal{P}$, so we want to show that the temporal operators behave nicely on paths. So we will prove the following lemma, that looks suspiciously close to our original Semantics (Definition 3.2).

Lemma 3.36 (Temporal Operators On Paths)**.** *Let $\mathcal{P}$ be a consistent and complete PNP and $\mathcal{P}_0, \mathcal{P}_1, \mathcal{P}_2, \cdots$ an infinite path in $\mathcal{G}_{\mathcal{P}}$, for $i \in \mathbb{N}$ and a a formula.*

1. *If $\bigcirc a \in \mathcal{F}_{\mathcal{P}_i}$, then $\bigcirc a \in pos(\mathcal{P}_i)$, if and only if $a \in pos(P_{i+1})$.*
2. *If $a \mathcal{W} b \in \mathcal{F}_{\mathcal{P}_i}$: $a \mathcal{W} b \in pos(\mathcal{P}_i)$, if and only if $b \in pos(P_k)$, for some $k \geq i$, implies that for every $j \in [i, k)$, $a \in pos(\mathcal{P}_j)$.*

Proof.

1. Assume that $\bigcirc a \in \mathcal{F}_{\mathcal{P}_i}$. If $\bigcirc a \in pos(\mathcal{P}_i)$ then $a \in pos(\sigma(\mathcal{P}_i))$ by the definition of σ and the completeness of $\mathcal{P}_i$. Then the definition of $\mathcal{G}_{\mathcal{P}}$ implies $a \in pos(\mathcal{P}_{i+1})$. Conversely, if $\bigcirc a \notin pos(\mathcal{P}_i)$, it must be that $\bigcirc a \in neg(\mathcal{P}_i)$. Then by the definition of σ and the completeness of $\mathcal{P}_i$, $a \in neg(\sigma(\mathcal{P}_i))$. Then $a \in neg(\mathcal{P}_{i+1})$ and by the disjointness of positive and negative sets (Lemma 3.22) we get $a \notin pos(\mathcal{P}_{i+1})$.
2. We prove each direction separately.

   ($\Rightarrow$) Assume $a \mathcal{W} b \in pos(\mathcal{P}_i)$, to show that there exists $k \geq i$ such that for every $j \in [i, k)$, $a \in pos(\mathcal{P}_j)$ and $b \in pos(P_k)$.

   By Taut and WkUntilUnroll $\vdash \widehat{\mathcal{P}_i} \to b$ or $\vdash \widehat{\mathcal{P}_i} \to (a \wedge \bullet(a \mathcal{W} b))$. If $\vdash \widehat{\mathcal{P}_i} \to b$, then we set $k = i$, and so vacuously $a \in pos(\mathcal{P}_j)$ for every $j \in [i, i) = \emptyset$ and $b \in \widehat{\mathcal{P}_i}$

   Otherwise, if $\vdash \widehat{\mathcal{P}_i} \to a \wedge \bullet(a \mathcal{W} b)$, then since $a \in \tau(a \mathcal{W} b)$, we have $a \in pos(P_i)$. We also know $a \mathcal{W} b \in pos(\mathcal{P}_{i+1})$ by the definition of σ. Continue inductively to see that either $a \in pos(\mathcal{P}_j)$ for all $j \geq i$, or there is some k such that $b \in pos(\mathcal{P}_j)$ and for every $i \leq j < k$, $a \in pos(\mathcal{P}_j)$.

   ($\Leftarrow$) Assume that $a \mathcal{W} b \notin pos(\mathcal{P}_i)$. Since $a \mathcal{W} b \in \mathcal{F}_{\mathcal{P}_i}$ it must be that $a \mathcal{W} b \in neg(\mathcal{P}_i)$. Then by Taut and WkUntilUnroll, $\vdash \widehat{\mathcal{P}_i} \to (\neg b \wedge \neg a)$ or $\vdash \widehat{\mathcal{P}_i} \to (\neg b \wedge \neg \bullet(a \mathcal{W} b))$.

   If $\vdash \widehat{\mathcal{P}_i} \to (\neg b \wedge \neg a)$, then we have $a, b \in neg(\mathcal{P}_i)$ which shows that neither is $a \in pos(\mathcal{P}_i)$ for all $k \geq i$ nor does there exist a $k \geq i$ where $b \in pos(\mathcal{P}_k)$ and $a \in pos(\mathcal{P}_j)$ for every $i \leq j < k$.

   Otherwise, we consider $\vdash \widehat{\mathcal{P}_i} \to \neg b \wedge \bigcirc\neg(a \mathcal{W} b)$. Consistency gives us that $b \in neg(\mathcal{P}_i)$ and the definition of σ tells us that

$a \mathcal{W} b \in neg(\mathcal{P}_{i+1})$, so we cannot set $k = i$. Continue inductively to conclude that $b \in neg(\mathcal{P}_i)$.

△

Some proofs using this method loosely refer to the proof graph structure as a *tree* [21], which is somewhat misleading since the graphs can have arbitrarily large cycles. The graph in Figure 3.4 of the proof graph constructed from the formula $\bigcirc \top$ already has a self-loop. However, for more complicated formulae, we can see more complicated graph structures. For example consider the graph constructed from the formula $\square((\bigcirc a) \vee b)$, in Figure 3.5. There are many cycles in this graph, such as $(\mathcal{Q}_3, \mathcal{Q}_4)$ and $(\mathcal{Q}_4, \mathcal{Q}_6, \mathcal{Q}_5, \mathcal{Q}_7)$.

Now we can show that every node $\mathcal{Q}$ that is not the end of time has a successor.

Lemma 3.37 (All Nodes Have Successors)**.** *Let $\mathcal{P}$ be a consistent and complete PNP, and $\mathcal{Q}_1, \cdots, \mathcal{Q}_n$ the nodes of $\mathcal{G}_{\mathcal{P}}$.*

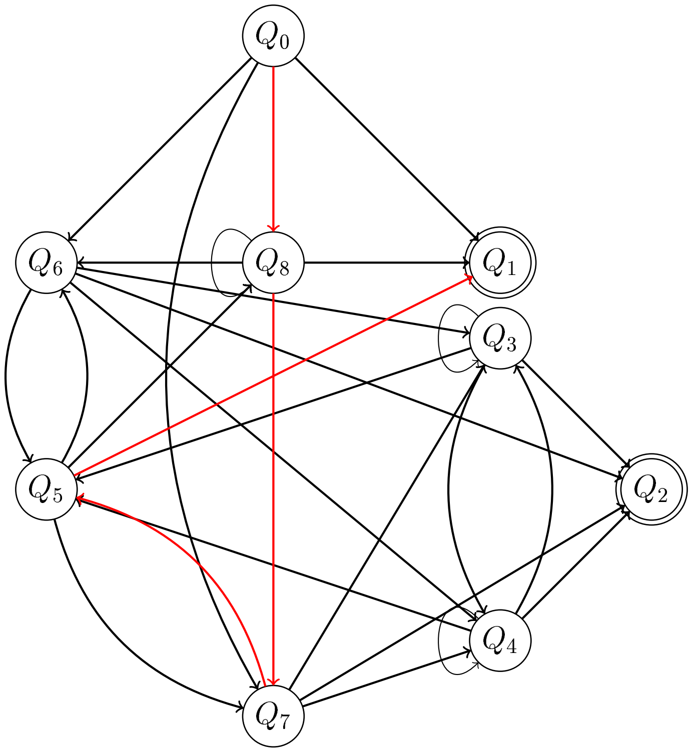
$$\vdash \bigvee_{i=1}^{n} \widehat{\mathcal{Q}_i} \rightarrow \bullet \bigvee_{i=1}^{n} \widehat{\mathcal{Q}}_i$$

Proof. By Lemma 3.25 and Lemma 3.33 (c.f. Lemma A.11). △

Reasoning about Paths

We want to find a path through this graph structure that we can use to construct a satisfying finite Kripke structure. When we're at some node $\mathcal{Q}$, it is connected to the consistent PNPs in $\mathsf{comps}(\sigma(\mathcal{Q}))$. If there are no consistent completions, then we know we are at the end of time. So we can maintain a path of consistent nodes, moving forwards in time with successive node. However, so far, we don't know that this path will end, or if it does where we can find that end point. We have the axiom FINITE which says that $\vdash \Diamond \mathsf{end}$, so we can be pretty confident that at some point every path will end. Let's call a path that does reach a node in which $\bigcirc \top$ is in the negative set a *terminal path.* In Figure 3.4, the terminal paths, were those that ended in $\mathcal{Q}_2$. In Figure 3.5, the red path is the chosen terminal path, but any path ending in one of the double-lined nodes is a terminal path.

Definition 3.38 (Terminal Path)**.** Given a consistent and complete PNP $\mathcal{P}$, and a graph $\mathcal{G}_{\mathcal{P}}$, a node $\mathcal{Z} \in \mathbf{V}(\mathcal{G}_{\mathcal{P}})$ is called *terminal* when $\bigcirc \top \in neg(\mathcal{Z})$. A path π is called *terminal* if it ends in a terminal node.



Node	Contents
Q_0	$(\{\Diamond\,\mathsf{end}, \Box(\bigcirc a \vee b)\}, \emptyset)$
Q_1	$(\{b, \top, \bigcirc a \vee b, \neg \bigcirc a, \mathsf{end}, \Box(\bigcirc a \vee b)\}, \{\bot, a, \bigcirc \top \vee \bot, \bigcirc a, \bigcirc \top, \Box \neg \mathsf{end}\})$
Q_2	$(\{a, b, \top, \bigcirc a \vee b, \neg \bigcirc a, \mathsf{end}, \Box(\bigcirc a \vee b)\}, \{\bot, \bigcirc \top \vee \bot, \bigcirc a, \bigcirc \top, \Box \neg \mathsf{end}\})$
Q_3	$(\{\bigcirc a \vee b, \bigcirc \top \vee \bot, \Diamond\,\mathsf{end}, \bigcirc a, \bigcirc \top, \Box(\bigcirc a \vee b)\}, \{\bot, b, \neg \bigcirc a, \mathsf{end}, \Box \neg \mathsf{end}\})$
Q_4	$(\{b, \bigcirc a \vee b, \neg \bigcirc a, \mathsf{end}, \Diamond\,\mathsf{end}, \Box(\bigcirc a \vee b)\}, \{\bot, \bigcirc \top \vee \bot, \bigcirc a, \bigcirc \top, \Box \neg \mathsf{end}\})$
Q_5	$(\{b, \bigcirc a \vee b, \bigcirc \top \vee \bot, \Diamond\,\mathsf{end}, \bigcirc a, \bigcirc \top, \Box(\bigcirc a \vee b)\}, \{\bot, \neg \bigcirc a, \mathsf{end}, \Box \neg \mathsf{end}\})$
Q_6	$(\{b, \bigcirc a \vee b, \bigcirc \top \vee \bot, \neg \bigcirc a, \Diamond\,\mathsf{end}, \bigcirc \top, \Box(\bigcirc a \vee b)\}, \{\bot, \mathsf{end}, \bigcirc a, \Box \neg \mathsf{end}\})$
Q_7	$(\{a, \top, \bigcirc a \vee b, \bigcirc \top \vee \bot, \bigcirc a, \bigcirc \top, \Box(\bigcirc a \vee b)\}, \{\bot, b, \neg \bigcirc a, \mathsf{end}, \Box \neg \mathsf{end}\})$
Q_8	$(\{a, b, \top, \bigcirc a \vee b, \bigcirc \top \vee \bot, \bigcirc a, \bigcirc \top, \Box(\bigcirc a \vee b)\}, \{\bot, \neg \bigcirc a, \mathsf{end}, \Box \neg \mathsf{end}\})$
Q_9	$(\{a, b, \top, \bigcirc a \vee b, \bigcirc \top \vee \bot, \neg \bigcirc a, \bigcirc \top, \Box(\bigcirc a \vee b)\}, \{\bot, \mathsf{end}, \bigcirc a, \Box \neg \mathsf{end}\})$
Q_{10}	$(\{\top, \bigcirc a \vee b, \bigcirc \top \vee \bot, \bigcirc a, \bigcirc \top, \Box(\bigcirc a \vee b)\}, \{\bot, a, b, \neg \bigcirc a, \mathsf{end}, \Box \neg \mathsf{end}\})$
Q_{11}	$(\{b, \top, \bigcirc a \vee b, \bigcirc \top \vee \bot, \bigcirc a, \bigcirc \top, \Box(\bigcirc a \vee b)\}, \{\bot, a, \neg \bigcirc a, \mathsf{end}, \Box \neg \mathsf{end}\})$
Q_{12}	$(\{b, \top, \bigcirc a \vee b, \bigcirc \top \vee \bot, \neg \bigcirc a, \bigcirc \top, \Box(\bigcirc a \vee b)\}, \{\bot, a, \mathsf{end}, \bigcirc a, \Box \neg \mathsf{end}\})$

Figure 3.5: The graph for $\Box((\bigcirc a) \vee b)$, with many cycles

Other proofs for LTL [21, 24] search for a different kind of path, called a *fulfilling path* which says that for every $i \in \mathbb{N}$, if $\Box\, a \in \mathit{neg}(\mathcal{P}_i)$, then $a \in \neg(\mathcal{P})$ for some $j \geq i$. We are using $\mathcal{W}$ instead of $\Box$ as our basic operator, so we will define an analogous definition for paths containing $a\, \mathcal{W}\, b$.

Definition 3.39 (Fulfilling Path)**.** Consider a path $\pi = \mathcal{P}_0, \mathcal{P}_1, \ldots$ in a graph $\mathcal{G}_{\mathcal{P}}$, where $\mathcal{P}$ is a consistent and complete PNP. A temporal formula $c \in \mathcal{F}_{\mathcal{P}_i}$ is called *fulfilled in* π when there exists a node $\mathcal{P}_j$ with $j \geq i$ such that $c \notin \tau(\sigma(\mathcal{P}_j))$. When the π is clear from context, we can just say that c is fulfilled. If every temporal term in π is fulfilled, call π a *fulfilling path.*

In fact, in LTL$_f$, we can demonstrate that every terminal path is a fulfilling path, thus greatly decreasing the proof obligation.

Lemma 3.40. *Every terminal path is a fulfilling path.*

Proof. In the terminal node, every as-yet unresolved temporal operator will be resolved in the terminal node (c.f. Lemma A.12). $\triangle$

Since we've shown that every terminal path is fulfilling, we need only to show that there exists a terminal path. Unfortunately this is not true for proof graphs in general, because they may me constructed without any information about the end of time. For example, we can have a PNP $\mathcal{Q} = (\{\Box\, a\}, \emptyset)$, which will never know anything about end or $\bigcirc \top$, because $\bigcirc \top \notin \tau(\mathcal{Q})$. To ensure that the tree structure knows about the end of time, we can insert the axiom $\vdash \Diamond\, \mathsf{end}$ into the positive set of the root PNP. This will make it so that every node is forced to consider whether to stick end into the positive or negative set; Lemma 3.41 demonstrates that end will end up in a positive set somewhere in the graph.

Lemma 3.41 (Terminal Path Existence)**.** *Let $\mathcal{P}$ be a consistent and complete PNP with $\Diamond\, \mathsf{end} \in \mathit{pos}(\mathcal{P})$. There is a terminal path in $\mathcal{G}_{\mathcal{P}}$.*

Proof. In fact, we only need to show that there exists a terminal node $\mathcal{Z}$ in the nodes of $\mathcal{G}_{\mathcal{P}}$, because $\mathcal{G}_{\mathcal{P}}$ is connected by definition. We proceed by induction on the size of $\mathcal{G}_{\mathcal{P}}$.

In the base case, the graph is the singleton node $\mathcal{P}$. The only way that a node has no successors is (by Lemma 3.26) when $\bigcirc \top \in \mathit{neg}(\mathcal{P})$, and so $\mathcal{Z} \equiv \mathcal{P}$.

Now, assume that we have a nontrivial $\mathcal{G}_{\mathcal{P}}$. Assume for the sake of contradiction that for every node $\mathcal{Q}$ of $\mathcal{G}_{\mathcal{P}}$, $\bigcirc \top \in \mathit{pos}(\mathcal{Q})$. Since $\Diamond\, \mathsf{end} \in \mathit{pos}(\mathcal{P})$, and every path out of $\mathcal{G}_{\mathcal{P}}$ has $\bigcirc\top \in pos(\mathcal{P})$, we know that for every node $\mathcal{Q}$ in $\mathcal{G}_{\mathcal{P}}$, $\Box\neg\neg\bigcirc\top \in neg(\mathcal{Q})$.

Let $\mathcal{Q}_1, \mathcal{Q}_2, \ldots, \mathcal{Q}_n$ be the nodes of $\mathcal{G}_{\mathcal{P}}$. Then since $\vdash \widehat{\mathcal{Q}_i} \to \bigcirc\top$ for every $i = 1, \ldots n$, we conclude $\vdash \bigvee_{i=1}^{n} \mathcal{Q}_i \to \bigcirc\top$.

Recall that $\vdash \bigvee_{i=1}^{n} \widehat{\mathcal{Q}_i} \to \bullet\bigvee_{i=1}^{n} \widehat{\mathcal{Q}_i}$ (Lemma 3.37). Now we can use Induction to conclude $\vdash \bigvee_{i=1}^{n} \widehat{\mathcal{Q}_i} \to \Box\bigcirc\top$. We have that $\mathcal{P}$ is a node of $\mathcal{G}_{\mathcal{P}}$, so $\mathcal{P} \in \{\mathcal{Q}_1, \cdots, \mathcal{Q}_n\}$, so we know that $\vdash \widehat{\mathcal{P}} \to \bigvee_{i=1}^{n} \widehat{\mathcal{Q}_i}$, and consequently $\vdash \widehat{\mathcal{P}} \to \Box\bigcirc\top$. However we assumed $\widehat{\mathcal{P}} \to \Diamond\,\mathsf{end}$, i.e. $\vdash \widehat{\mathcal{P}} \to \neg\Box\bigcirc\top$, contradicting the assumption that every node $\mathcal{Q}$ of $\mathcal{G}_{\mathcal{P}}$ has $\bigcirc\top \in pos(\mathcal{Q})$. Therefore, there exists a node $\mathcal{Z}$ such that $\bigcirc\top \in neg(\mathcal{Z})$. △

The Proof

Now we are finally in a position to prove the satisfiability result (Theorem 3.42) and use it to derive the completeness theorem (Theorem 3.43).

Theorem 3.42 (Satisfiability Theorem for LTL$_f$). *For any consistent PNP $\mathcal{P}$, the formula $\widehat{\mathcal{P}}$ is satisfiable.*

Proof. $\mathcal{P}' = (\{\Diamond\,\mathsf{end}\} \cup pos(\mathcal{P}), neg(\mathcal{P}))$ is a consistent PNP, $\mathcal{P}^*$ is a completion of $\mathcal{P}'$. And $P_0, P_1, \cdots, P_n$ a complete, finite, rooted path of length n (by Lemma 3.41) in $\mathcal{G}_{\mathcal{P}*}$. Define $K^n = (\eta_0, \eta_1, \cdots, \eta_n)$ by

$$\begin{aligned} \eta_i(v) &= \textbf{true} && \text{if } v \in \mathcal{F}_i^+ \\ \eta_i(v) &= \textbf{false} && \textit{otherwise} \end{aligned}$$

We want to show that $K_0^n(\widehat{\mathcal{P}}) = \textbf{true}$. This is equivalent to showing that $K_0^n(\widehat{\mathcal{P}'}) = \textbf{true}$, which is also equivalent to showing that $K_0^n(\widehat{P_0}) = \textbf{true}$. Proving this is hard, so we will generalize the hypothesis making it easier to prove. We will show that for PNP P_i, and $f \in \mathcal{F}_{P_i}$, then $K_i^n(f) = \textbf{true}$ iff $f \in pos(P_i)$, and $K_i^n(f) = \textbf{false}$ otherwise.

We proceed by induction on the derivation of an arbitrary formula f. Here show only the interesting cases corresponding to the operators $\to$ and $\mathcal{W}$. For the full proof see Lemma A.14

($f = a \to b$). Since $\mathcal{P}_i$ is a complete PNP, we have that $a, b \in \mathcal{F}^+ \cup \mathcal{F}^-$. The inductive hypothesis allows us to conclude that $K_i^n(a) = \textbf{false}$ iff

$a \notin \mathcal{F}_i^+$. It also allows the conclusion $K_i^n(b) = \textsf{true}$ iff $b \in \mathcal{F}_i^+$. Then, $K_i^n(a \to b) = \textsf{true}$ if and only if $K_i^n(a) = \textsf{false}$ or $K_i^n(b) = \textsf{true}$, we can apply the inductive hypotheses to get $a \in \mathcal{F}_i^-$ or $b \in \mathcal{F}_i^+$. Then by Lemma 3.22, we can conclude that $a \to b \in \mathcal{F}_i^+$.

$(f = a \mathcal{W} b)$. Here we dont need to break into cases, we can prove this directly. $K_i^n(a \mathcal{W} b) = \textsf{true}$ if and only if there exists $k \in [i, n]$ such that for all $j \in [i, k)$, $K_j^n(a) = \textsf{true}$ and $K_k^n(b) = \textsf{true}$. using the inductive hypothesis, we know this statement holds if and only if there exists $k \in [i, n]$ such that for all $j \in [i, k)$, $a \in pos(\mathcal{P}_j)$ and $b \in pos(\mathcal{P}_k)$. Then by Lemma 3.36 the previous statement holds if and only if $a \mathcal{W} b \in pos(\mathcal{P}_i)$.

△

Theorem 3.43 (Weak Completeness Theorem for LTL$_f$)**.** *Given a finite set of formulae $\mathcal{F} = a_1, a_2, \cdots a_n$ and a formula b, if $\mathcal{F} \vDash a$, then $\mathcal{F} \vdash a$.*

Proof. Firstwe prove the claim for $\mathcal{F} = \emptyset$. If $\vDash a$, then $\neg a$ is not satisfiable by Theorem 3.18. Consequently the PNP $(\emptyset, \{a\})$ is inconsistent by Theorem 3.42. By the definition of consistency, we get $\vdash \neg\neg a$ which gives $\vdash a$ using TAUT. Let us now show that a proof of $\vDash a$ implies $\vdash a$ is sufficient.

Rewrite to be by induction on $n = |F|$?

Let $\mathcal{F} = a_1, \cdots a_n$ for $n > 0$. We then have $a_1, \cdots a_n \vDash a$. Then by Theorem 3.17 we have $a_1, \cdots a_{n-1} \vDash \Box a_n \to a$. We can apply Theorem 3.17 $n - 1$ more times to get $\vDash \Box a_1 \to \cdots \to \Box a_n \to a$, which, as proved above, gives $\vdash \Box a_1 \to \cdots \to \Box a_n \to a$. Then by applying Theorem 3.21 n times we get $\mathcal{F} \vdash a$. △

3.5 A Decision Procedure for LTL$_f$

In Section 3.2, we defined the terms *valid* and *satisfiable* with respect to formulae of LTL$_f$. These are great as mathematical definitions and work well when we can simply assume the validity or satisfiability of a formula. We now want a way to *decide* whether a formula is satisfiable, valid, or otherwise.

Now, how might we fix this? We need to come up with an algorithm to decide whether a given formula is satisfiable, valid, or unsatisfiable. This is equivalent to deciding whether, for a given formula, *there exists* a satisfying

K^n, *every* K^n is satisfying, or no K^n is satisfying, respectively. Notice that if the negation of a formula is unsatisfiable, then that formula is valid, so our algorithm will only need to detect whether a given formula is satisfiable.

The decision procedure will look very much like a traversal of $\mathcal{G}_{\mathcal{P}}$ for a PNP $\mathcal{P}$. The difference is that now we are not allowed to know anything about the consistency of a node, a central piece of knowledge in the contstruction of $\mathcal{G}_{\mathcal{P}}$. If we knew whether a PNP $\mathcal{P}$ was consistent, we would then know that $\widehat{\mathcal{P}}$ is satisfiable by soundness (Theorem 3.20). Instead, for a given PNP, we will create successor nodes that are be implied by the current node, i.e. if some node $\mathcal{Q}$ evaluates to **true**, then one of its successors also evaluates to **true**. We will call our structure a *tableau* (Definition 3.46). To make a tableau, we are constructively exploring the potential truth values of the subformulae in a node $\mathcal{Q}$ to determine if it is satsifiable, whereas in the proof graph, we are assuming consistency of each node to prove satisfiability. We have no way of knowing anything about the consistency or satisfiability of a given node in an arbitrary tableau.

As an example, consider a node $\mathcal{P}$ with $a \to b \in pos(\mathcal{P})$. We want to perfom a case analysis on a and b, so we create 2 successors: a left successor with $\neg a$ in the positive set, and a right successor with b in the positive set. Critically, we remove $a \to b$ from the positive sets of the successors, since both $\vdash \neg a \to (a \to b)$ and $\vdash b \to (a \vee b)$ by TAUT. Thus, removing $a \to b$ has the nice feature that $|\tau(\mathcal{P})|$ is decreasing.

The tricky case to consider, as we might expect, is what to do when $\bigcirc \top \in neg(\mathcal{P})$. At this point we want to try and end time, so we will define a transformation on PNPs that removes all temporal information. For variables, implication and the $\bot$ symbol we can just remove temporal operators for any subformulae that may occur. However, for operators like $\mathcal{W}$ and $\bigcirc$ we need to take some action.

As in the semantics and proof theory, if we are in the last state and see a formula matching the pattern $\bigcirc a$, we simply transform it into $\bot$. If we see a formula $a \mathcal{W} b$, we will need to do some more work. Recall the WKUNTILUNROLL axiom which says that $a \mathcal{W} b \leftrightarrow b \vee a \wedge \bullet(a \mathcal{W} b)$. Since we are at the end of time, the term $\bullet(a \mathcal{W} b)$ just becomes $\top$ and falls away, giving $a \vee b$. Then since a and b might themselves contain temporal operators, we will recursively continue the transformation on those formulae. This is defined formally as the drop function below:

Definition 3.44 (drop Function)**.** Define $\mathsf{drop} : \mathrm{LTL}_f \to \mathrm{LTL}_f$ below

$$\mathsf{drop}(c) = \begin{cases} \bot & \text{if } c \equiv \bot \\ \bot & \text{if } c \equiv \bigcirc a \\ v & \text{if } v \in \mathbf{V} \\ \mathsf{drop}(a) \vee \mathsf{drop}(b) & \text{if } c \equiv a \mathcal{W} b \\ \mathsf{drop}(a) \to \mathsf{drop}(b) & \text{if } c \equiv a \to b \end{cases}$$

We can lift drop to sets of formulae, setting

$$\mathsf{drop}(\mathcal{F}) \triangleq \bigcup_{c \in Form} \mathsf{drop}(c).$$

Lemma 3.45. *For every formula c and every finite Kripke structure K^n,*

$$K^n_n(c) = K^n_n(\mathsf{drop}(c))$$

Proof. Let c be an arbitrary formula, K^n an arbitrary Kripke structure. Show that $K^n_n(c) = K^n_n(c)$. By induction on the structure of c (c.f. Lemma A.15). $\triangle$

Now that we have a schema for dealing with syntactic operators, and a way for dealing with the end of time, we can formally define our construction of a tableau.

Definition 3.46 (Tableau)**.** Given a PNP $\mathcal{P}$ we construct a *tableau for* $\mathcal{P}$, written $\mathcal{T}_{\mathcal{P}}$. Let the PNP $(\{\Diamond \mathsf{end}\} \cup pos(\mathcal{P}), neg(\mathcal{P}))$, be the root node of $\mathcal{T}_{\mathcal{P}}$. For a given node $\mathcal{Q}$ in $\mathcal{T}_{\mathcal{P}}$, construct its successors using one of the following rules. If more than one rule is relevant, apply the one occuring earlier in the list.

($\bot$) If $\bot \in pos(\mathcal{Q})$ or $pos(\mathcal{Q}) \cap neg(\mathcal{Q}) \neq \emptyset$, there are no successors.

($\to^+$) If $a \to b \in pos(\mathcal{Q})$, the successors are:

- $((pos(\mathcal{Q})/\{a \to b\}) \cup \{b\}, neg(\mathcal{Q}))$, and
- $((pos(\mathcal{Q})/\{a \to b\}), neg(\mathcal{Q}) \cup \{a\})$

($\to^-$) If $a \to b \in neg(\mathcal{Q})$, the successor is:

- $(pos(\mathcal{Q}) \cup \{a\}, (neg(\mathcal{Q})/(a \to b) \cup \{b\})$

($\mathcal{W}^+$) If $a \mathcal{W} b \in pos(\mathcal{Q})$, the successors are:

- $((pos(\mathcal{Q})/\{a \mathcal{W} b\}) \cup \{b\}, neg(\mathcal{Q}))$, and
- $((pos(\mathcal{Q})/\{a \mathcal{W} b\}) \cup \{a, \bullet(a \mathcal{W} b)\}, neg(\mathcal{Q}))$

($\mathcal{W}^-$) If $a \mathcal{W} b \in neg(\mathcal{Q})$, the successors are:

- $(pos(\mathcal{Q}), (neg(\mathcal{Q})/\{a \mathcal{W} b\}) \cup \{a, b\})$, and
- $(pos(\mathcal{Q}) \cup \{\bigcirc \neg(a \mathcal{W} b)\}, (neg(\mathcal{Q})/\{a \mathcal{W} b\}) \cup \{b\})$

(end) If all of the elements of $\mathcal{F}_{\mathcal{Q}}$ are either the symbol $\bot$, a variable, or of the form $\bigcirc a$ for any given a, and both $\bigcirc \top \in neg(\mathcal{Q})$ and $\sigma_1^+(\mathcal{Q}) = \emptyset$, the successor is

- $(\mathsf{drop}(pos(\mathcal{Q})), \mathsf{drop}(neg(\mathcal{Q})) \cup \{\bigcirc \top\})$

($\bigcirc$) If all of the elements of $\mathcal{F}_{\mathcal{Q}}$ are either the symbol $\bot$, a variable, or of the form $\bigcirc a$ for any given a, the successor is

- $(\sigma_1^+(\mathcal{Q}), \sigma_4^-(\mathcal{Q}))$

If more than one rule applies at any given state, only the rule that comes first in the above list. Following this additional rule has the effect of expanding all of the classical logic in the current time step before moving forward to the next one. In fact, the number of time steps in a path from the root node of the tableau can be measured by the number of times that the ($\bigcirc$) rule applies (Definition 3.55).

Notice that this tableau is slightly different from the graphs that we saw in Definition 3.34. For a PNP $\mathcal{P}$ and its graph $\mathcal{G}_{\mathcal{P}}$, the step from a node A to a node B represented a step forwards in time, and so in a path, we could simply map valuation functions to nodes. With respect to tableaux, however, the successor relationship does not imply a time step. Only the application of the ($\bigcirc$) rule measures time steps; the time in between is an exploration (akin to the τ function) of the formulae in the current time step. Let's see that these rules are sensible:

Lemma 3.47 (Rules are Sound)**.** *Let $\mathcal{T}_{\mathcal{P}}$ be a tableau for a PNP $\mathcal{P}$, and $\mathcal{Q}$ be some node of $\mathcal{T}_{\mathcal{P}}$. For all temporal structures K and all $i \in \mathbb{N}$:*

a) If $(\bigcirc)$ *does not apply, then* $K_i^n(\widehat{\mathcal{Q}}) = K_i^n(\widehat{\mathcal{Q}'})$ *for some successor* $\mathcal{Q}'$ *of* $\mathcal{Q}$*.*

b) If $(\bigcirc)$ *applies, then* $K_i^n(\widehat{\mathcal{Q}}) = K_{i+1}^n(\widehat{\mathcal{Q}'})$ *for the successor* $\mathcal{Q}'$ *of* $\mathcal{Q}$*. Also,* $\widehat{\mathcal{Q}'}$ *is satisfiable only if* $\widehat{\mathcal{Q}}$ *is satsifiable.*

Proof. By cases on the rule applied (c.f. Lemma A.16). △

Similar to the proof of completeness (Theorem 3.43), we need to find something akin to a terminal path to construct a satsifable finite Kripke structure. We define it slightly differently, because in the last state, we still need to examine those non-temporal conditions that hold in the final state. Define a *terminating* node, is a node in a tableau $\mathcal{T}_\mathcal{P}$ that has $\bigcirc\top$ in the negative set, or is a successor of a node $\mathcal{Z}$ that has $\bigcirc\top$ in the negative set. For simplicity, we will simply continue to add $\bigcirc\top$ to the negative sets of $\mathcal{Z}$'s successors.

Definition 3.48 (Terminating Node in a Tableau)**.** A node $\mathcal{Z}$ is *terminating* in a tableau $\mathcal{T}_\mathcal{P}$ if $\bigcirc\top \in neg(\mathcal{Z})$, and every element in $\mathcal{F}_\mathcal{Z}/\{\bigcirc\top\}$ is $\bot$ or a variable.

Definition 3.49 (Terminating Path)**.** A *terminating path* in $\mathcal{T}_\mathcal{P}$ is a path that ends in a terminating node with no successors.

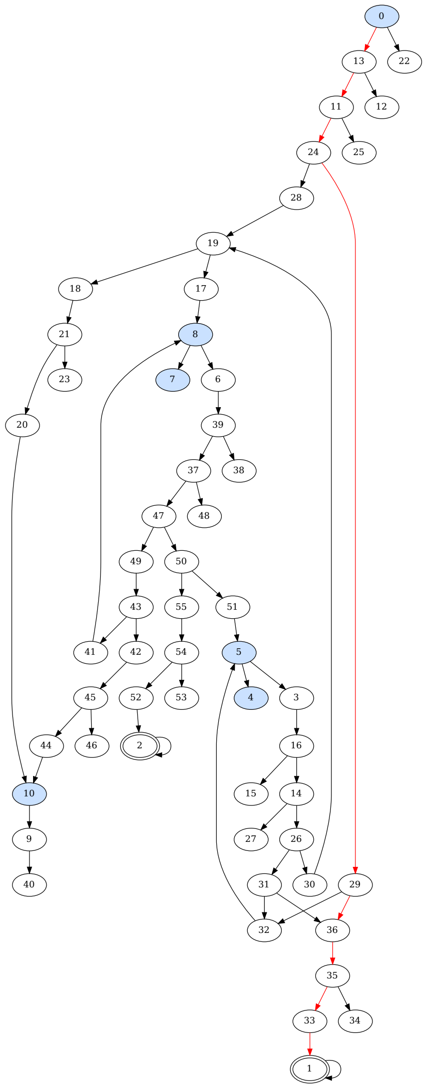
Let's look at a simple example of this construction. Most notably, consider the tableau constructed from the PNP $(\{\Diamond\,\mathsf{end}, \Box((\bigcirc a) \vee b)\}, \emptyset)$, which is the same PNP as in the root node of the proof graph in Figure 3.5. In Figure 3.6 we observe a much larger graph, because eacu successor in $\textbf{true}_\mathcal{P}$ represents only a single step of logical unfolding, but σ represents a whole time step. Notice that there is a terminating path from node $\mathcal{Q}_0$ to node $\mathcal{Q}_1$, this is denoted by the the red path.

Definition 3.50 (Closed Nodes)**.** A node $\mathcal{Q}$ in a tableau $\mathcal{T}_\mathcal{P}$ for a PNP $\mathcal{P}$ is closed if:

$(C1)$ The $(\bot)$ rule applies to $\mathcal{Q}$, or

$(C2)$ All the successors of $\mathcal{Q}$ are closed, or

$(C3)$ All terminating paths from $\mathcal{Q}$ have at least one closed node.



Node	Contents
Q_0	$(\{\Diamond\,\mathsf{end}, \Box(\bigcirc a \vee b)\}, \emptyset)$
Q_1	$(\{b\}, \{\bot, \bigcirc \top\})$
Q_2	$(\{a, b\}, \{\bot, \bigcirc \top\})$
Q_3	$(\emptyset, \{\neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_4	$(\{\bot\}, \{\neg \Box(\bigcirc a \vee b)\})$
Q_5	$(\{\Diamond\,\mathsf{end}\}, \{\neg \Box(\bigcirc a \vee b)\})$
Q_6	$(\{a\}, \{\neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_7	$(\{\bot, a\}, \{\neg \Box(\bigcirc a \vee b)\})$
Q_8	$(\{a, \Diamond\,\mathsf{end}\}, \{\neg \Box(\bigcirc a \vee b)\})$
Q_9	$(\{a, \Box(\bigcirc a \vee b)\}, \{\bot, \top\})$
Q_{10}	$(\{a\}, \{\top, \neg \Box(\bigcirc a \vee b)\})$
Q_{11}	$(\{\bigcirc a \vee b, \bullet \Box(\bigcirc a \vee b)\}, \{\Box \neg\mathsf{end}\})$
Q_{12}	$(\{\bot\}, \{\Box \neg\mathsf{end}\})$
Q_{13}	$(\{\Box(\bigcirc a \vee b)\}, \{\Box \neg\mathsf{end}\})$
Q_{14}	$(\{\bigcirc a \vee b, \bullet \Box(\bigcirc a \vee b)\}, \{\bot, \Box \neg\mathsf{end}\})$
Q_{15}	$(\{\bot\}, \{\bot, \Box \neg\mathsf{end}\})$
Q_{16}	$(\{\Box(\bigcirc a \vee b)\}, \{\bot, \Box \neg\mathsf{end}\})$
Q_{17}	$(\{\bigcirc a, \bigcirc \Diamond\,\mathsf{end}\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{18}	$(\{\bigcirc a\}, \{\bot, \bigcirc \top \vee \bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{19}	$(\{\bigcirc a\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{20}	$(\{\bigcirc a\}, \{\bot, \bigcirc \top, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{21}	$(\{\mathsf{end}, \bigcirc a\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{22}	$(\{\bot, \Box(\bigcirc a \vee b)\}, \emptyset)$
Q_{23}	$(\{\bot, \bigcirc a\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{24}	$(\{\bigcirc a \vee b\}, \{\bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{25}	$(\{\bot, \bigcirc a \vee b\}, \{\Box \neg\mathsf{end}\})$
Q_{26}	$(\{\bigcirc a \vee b\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{27}	$(\{\bot, \bigcirc a \vee b\}, \{\bot, \Box \neg\mathsf{end}\})$
Q_{28}	$(\emptyset, \{\neg \bigcirc a, \bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{29}	$(\{b\}, \{\bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{30}	$(\emptyset, \{\bot, \neg \bigcirc a, \bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{31}	$(\{b\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{32}	$(\{b, \bigcirc \Diamond\,\mathsf{end}\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{33}	$(\{b\}, \{\bot, \bigcirc \top, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{34}	$(\{\bot, b\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{35}	$(\{b, \mathsf{end}\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{36}	$(\{b\}, \{\bot, \bigcirc \top \vee \bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{37}	$(\{a, \bigcirc a \vee b, \bullet \Box(\bigcirc a \vee b)\}, \{\bot, \Box \neg\mathsf{end}\})$
Q_{38}	$(\{\bot, a\}, \{\bot, \Box \neg\mathsf{end}\})$
Q_{39}	$(\{a, \Box(\bigcirc a \vee b)\}, \{\bot, \Box \neg\mathsf{end}\})$
Q_{40}	$(\{\bot, a, \Box(\bigcirc a \vee b)\}, \{\bot\})$
Q_{41}	$(\{a, \bigcirc a, \bigcirc \Diamond\,\mathsf{end}\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{42}	$(\{a, \bigcirc a\}, \{\bot, \bigcirc \top \vee \bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{43}	$(\{a, \bigcirc a\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{44}	$(\{a, \bigcirc a\}, \{\bot, \bigcirc \top, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{45}	$(\{a, \mathsf{end}, \bigcirc a\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{46}	$(\{\bot, a, \bigcirc a\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{47}	$(\{a, \bigcirc a \vee b\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{48}	$(\{\bot, a, \bigcirc a \vee b\}, \{\bot, \Box \neg\mathsf{end}\})$
Q_{49}	$(\{a\}, \{\bot, \neg \bigcirc a, \bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{50}	$(\{a, b\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b), \Box \neg\mathsf{end}\})$
Q_{51}	$(\{a, b, \bigcirc \Diamond\,\mathsf{end}\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{52}	$(\{a, b\}, \{\bot, \bigcirc \top, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{53}	$(\{\bot, a, b\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{54}	$(\{a, b, \mathsf{end}\}, \{\bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$
Q_{55}	$(\{a, b\}, \{\bot, \bigcirc \top \vee \bot, \bigcirc \neg \Box(\bigcirc a \vee b)\})$

Figure 3.6: The tableau constructed from the formula $\Box((\bigcirc a) \vee b)$

In Figure 3.7 the tableau is unsuccessful because $\mathcal{Q}_0$ is closed. Which is a result of $\mathcal{Q}_8$ and $\mathcal{Q}_{14}$ being closed. $\mathcal{Q}_{14}$ is closed because $\bot$ is in the positive set, and $\mathcal{Q}_8$ is closed because every successive node is also closed (something that can be verified by carefully checking the tableau).

Definition 3.51 (Successful Tableau)**.** For a PNP $\mathcal{P}$, the tableau $\mathcal{T}_{\mathcal{P}}$ is *unsuccessful* when the root node is closed, otherwise, it is *successful.*

The tableau in Figure 3.6 is a good example of a *successful* tableau, since there is the red path from $\mathcal{Q}_0$ to $\mathcal{Q}_1$ that specifies a terminating path, which contains no closed nodes (Lemma 3.53). On the other hand, the tableau constructed from $\Box \bigcirc \top$ in Figure 3.7 is closed, since all of the children of $\mathcal{Q}_0 = (\{\Diamond \mathsf{end}, \Box \bigcirc \top\}, \emptyset)$. This makes sense since the formula used to construct this PNP, $\Box \bigcirc \top$, is not satisfiable, since it is indirect contradiction with the axiom FINITE.

Definition 3.52 (Tableau Path)**.** For a tableau $\mathcal{T}_{\mathcal{P}}$ for a PNP $\mathcal{P}$, a finite Kripke structure K^n, and a node $\mathcal{Q}$ in $\mathcal{T}_{\mathcal{P}}$, we inductively define the path $\pi^{K^n}(\mathcal{Q})$, where $\mathcal{Q}_i$ represents the ith node in the path $\pi^{K^n}(\mathcal{Q})$.

1. $\mathcal{Q}_0 = \mathcal{Q}$.

2. If $\mathcal{Q}_i$ has no successor node, then $\mathcal{Q}_i$ is the last node.

3. If there is only a single successor to $\mathcal{Q}_i$, call it $\mathcal{Q}'$, let $\mathcal{Q}_{i+1} = \mathcal{Q}$

4. If $\mathcal{Q}_i$ has two successor nodes $\mathcal{Q}_l$ and $\mathcal{Q}_r$, then if $K_{cnt(i)}(\widehat{\mathcal{Q}_l}) = \textbf{true}$, set $\mathcal{Q}_{i+1} = \mathcal{Q}_l$ else set $\mathcal{Q}_{i+1} = \mathcal{Q}_l$.

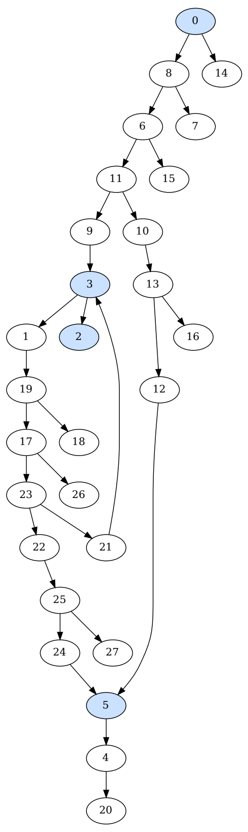
Lemma 3.53 (Tableau Paths Not Closed)**.** *Let $\mathcal{Q}$ be a node in a tableau and K^n a finite Kripke structure such that $K^n_0(\widehat{\mathcal{Q}}) = \textbf{true}$. Consider a path $\pi^{K^n}(\mathcal{Q}) = \mathcal{Q}_0, \mathcal{Q}_1, \ldots, \mathcal{Q}_m$, then*

(a) $\pi^{K^n}(\mathcal{Q})$ does not contain any closed node, and

(b) For every $i = 0, \ldots, m$, then $K^n_{cnt(i)}(\widehat{\mathcal{Q}_i}) = \textbf{true}$.

Proof. See Lemma A.17 for proof. △

Lemma 3.54 (Terminating Path Existence)**.** *Every successful tableau $\mathcal{T}_{\mathcal{P}}$ for a PNP $\mathcal{P}$ contains a terminating path.*



Node	Contents
Q_0	$(\{\Diamond\,\mathsf{end}, \Box\bigcirc a\}, \emptyset)$
Q_1	$(\{a\}, \{\neg\Box\bigcirc a, \Box\neg\mathsf{end}\})$
Q_2	$(\{\bot, a\}, \{\neg\Box\bigcirc a\})$
Q_3	$(\{a, \Diamond\,\mathsf{end}\}, \{\neg\Box\bigcirc a\})$
Q_4	$(\{a, \Box\bigcirc a\}, \{\bot, \top\})$
Q_5	$(\{a\}, \{\top, \neg\Box\bigcirc a\})$
Q_6	$(\{\bullet\Box\bigcirc a, \bigcirc a\}, \{\Box\neg\mathsf{end}\})$
Q_7	$(\{\bot\}, \{\Box\neg\mathsf{end}\})$
Q_8	$(\{\Box\bigcirc a\}, \{\Box\neg\mathsf{end}\})$
Q_9	$(\{\bigcirc a, \bigcirc\Diamond\,\mathsf{end}\}, \{\bot, \bigcirc\neg\Box\bigcirc a\})$
Q_{10}	$(\{\bigcirc a\}, \{\bot, \bigcirc\top\vee\bot, \bigcirc\neg\Box\bigcirc a\})$
Q_{11}	$(\{\bigcirc a\}, \{\bigcirc\neg\Box\bigcirc a, \Box\neg\mathsf{end}\})$
Q_{12}	$(\{\bigcirc a\}, \{\bot, \bigcirc\top, \bigcirc\neg\Box\bigcirc a\})$
Q_{13}	$(\{\mathsf{end}, \bigcirc a\}, \{\bot, \bigcirc\neg\Box\bigcirc a\})$
Q_{14}	$(\{\bot, \Box\bigcirc a\}, \emptyset)$
Q_{15}	$(\{\bot, \bigcirc a\}, \{\Box\neg\mathsf{end}\})$
Q_{16}	$(\{\bot, \bigcirc a\}, \{\bot, \bigcirc\neg\Box\bigcirc a\})$
Q_{17}	$(\{a, \bullet\Box\bigcirc a, \bigcirc a\}, \{\bot, \Box\neg\mathsf{end}\})$
Q_{18}	$(\{\bot, a\}, \{\bot, \Box\neg\mathsf{end}\})$
Q_{19}	$(\{a, \Box\bigcirc a\}, \{\bot, \Box\neg\mathsf{end}\})$
Q_{20}	$(\{\bot, a, \Box\bigcirc a\}, \{\bot\})$
Q_{21}	$(\{a, \bigcirc a, \bigcirc\Diamond\,\mathsf{end}\}, \{\bot, \bigcirc\neg\Box\bigcirc a\})$
Q_{22}	$(\{a, \bigcirc a\}, \{\bot, \bigcirc\top\vee\bot, \bigcirc\neg\Box\bigcirc a\})$
Q_{23}	$(\{a, \bigcirc a\}, \{\bot, \bigcirc\neg\Box\bigcirc a, \Box\neg\mathsf{end}\})$
Q_{24}	$(\{a, \bigcirc a\}, \{\bot, \bigcirc\top, \bigcirc\neg\Box\bigcirc a\})$
Q_{25}	$(\{a, \mathsf{end}, \bigcirc a\}, \{\bot, \bigcirc\neg\Box\bigcirc a\})$
Q_{26}	$(\{\bot, a, \bigcirc a\}, \{\bot, \Box\neg\mathsf{end}\})$
Q_{27}	$(\{\bot, a, \bigcirc a\}, \{\bot, \bigcirc\neg\Box\bigcirc a\})$

Figure 3.7: Unsuccessful Tableau evaluating the formula $\Box\bigcirc\top$.

Proof. Let $\mathcal{T}_{\mathcal{P}}$ be a successful tableau for a PNP $\mathcal{P}$. We need to find a path to a non-closed terminating node that has no successors. We will simply present a way to construct a terminating path that is not closed. Let $\pi_0 = (\{\Diamond \mathsf{end}\} \cup pos(\mathcal{P}), neg(\mathcal{P}))$. Then, note that π_0 is not closed, since is only node is the root node of a successful tree.

Now consider the general case. Given an arbitrary $\pi_i = P_0, \dots P_k$ that has already been defined, and no node in π_i is closed. Hence, by the definition of closed, P_k as at least one successor that is not closed. Append one of these successors P' to π_i to derive π_{i+1}.

Continue until there are no more successors. Now we must demonstrate that the output of this procedure is terminating. Let $\mathcal{Z}$ be the final node. If $\mathcal{Z}$ is not terminating, then $\Box \neg \mathsf{end}$ must be in $neg(\mathcal{Z})$ — and so $\mathcal{Z}$ has a successor. Sincer $\mathcal{Z}$ is not closed, it has a successor that it not closed, contradicting the assumption that $\mathcal{Z}$ is final in the path. So, $\mathcal{Z}$ must be terminating. △

We need to come up with a way to convert between the index of a path in a tableau, and the index of an η-function in a Kripke structure. To do this, we will use two functions, *idx* and *cnt*. These will be pseudo inverses, where $cnt(idx(i)) = i$ but $idx(cnt(i)) \geq i$. The intuition for this is that *cnt* is counting the number of η functions, and *idx* gives the index of the node in the path. We define these formally below:

Definition 3.55 (Indexing functions)**.** Given a path $\pi = P_0, P_1, P_2, \dots$ in a tableau, define the monotonic function

$$cnt_\pi(i) = |\{j < i \mid (\bigcirc) \text{ applies to } P_j\}|$$

and

$$idx_\pi(i) = \max\{i \in \mathbb{N} \mid cnt(i) = k\}$$

We omit the π and just write *cnt* and *idx* when the path is obvious from context.

We use these to define the translation bewteen paths and Kripke structures.

Lemma 3.56 (Path Modeling)**.** *Given a rooted, finite, terminating path* $\pi = P_0, P_1, P_2, \dots, P_n$*, in a tableau* $\mathcal{T}_{\mathcal{P}}$ *for some PNP* $\mathcal{P}$*. If* K^n *is a temporal* *structure such that for every $v \in \boldsymbol{V}$ and $i \in \mathbb{N}$,*

$$v \in pos(P_{idx(i)}) \Rightarrow \eta_i(v) = \textsf{\textbf{true}}$$
$$v \in neg(P_{idx(i)}) \Rightarrow \eta_i(v) = \textsf{\textbf{false}},$$

then for every $v \in \boldsymbol{V}$ and $i \in \mathbb{N}$,

$$a \in pos(P_i) \Rightarrow K_{cnt(i)}(a) = \textsf{\textbf{true}}$$
$$a \in neg(P_i) \Rightarrow K_{cnt(i)}(a) = \textsf{\textbf{false}}.$$

Proof. Let $\mathcal{P}$ be a PNP, and $\mathcal{T}_{\mathcal{P}}$ be a tableau. Let $P_0, P_1, \ldots, P_n$ be a terminating path such that $P_0 = \mathcal{P}$. Let K^n be defined such that for every $i \in \{1, \ldots, n\}$ and variable $v \in \mathbf{V}$,

$$v \in pos(P_{idx(i)}) \Rightarrow \eta_i(v) = \textsf{\textbf{true}}$$
$$v \in neg(P_{idx(i)}) \Rightarrow \eta_i(v) = \textsf{\textbf{false}}.$$

Consider an arbitrary node P_i for some $i \in \{1, \ldots, n\}$, and a formula $a \in \mathcal{F}_{P_i}$. Show that $a \in pos(P_i)$ means that $K_{cnt(i)}(a) = \textsf{\textbf{true}}$ and $a \in neg(P_i)$ implies $K_{cnt(i)}(a) = \textsf{\textbf{false}}$. We will proceed by induction on the structure of the formula a, showing only the interesting cases, the remaining ones can be seen in the proof of Lemma A.18:

Case 1: $(a \equiv b \to c)$. Assume $a \in pos(P_i)$. At the current "time slice" we have the nodes $P_i, \ldots, P_{idx(cnt(i))}$. Since node $P_{idx(cnt(i))+1}$ represents an application of (end) or ($\bigcirc$), we know that at some index $i \le j < idx(cnt(i))$, the rule $(\to^+)$ is applied to a. By construction, it follows that $b \in neg(P_{j+1})$ or $c \in pos(P_{j+1})$. The induction hypothesis gives that $K^n_{cnt(j)}(A) = \textsf{\textbf{true}}$, and that $K^n_{cnt(j)}(A) = \textsf{\textbf{true}}$. This gives us $K^n_{cnt(j)}(A) = \textsf{\textbf{true}}$. Note that since $i \le j < st(cnt(i))$, we know that $cnt(j) = cnt(i)$, and so we conclude that $K^n_{cnt(i)}(A) = \textsf{\textbf{true}}$.

Assume, contrarily, that $a \in neg(P_i)$. Consider the same "time slice" portion of the path as above, namely $P_i, \ldots, P_{idx(cnt(i))}$. By the same analysis there must be some index $i \le j \le idx(cnt(i))$ such that $(\to^-)$ is applied to a in P_j. This means that $b \in pos(P_{j+1})$ and $c \in neg(P_{j+1})$. The inductive hypothesis gives that $K^n_{cnt(j)}(a) = \textsf{\textbf{true}}$ and $K^n_{cnt(j)}(c) = \textsf{\textbf{false}}$. The definition of K^n and the fact that $cnt(i) = cnt(j)$ means that $K^n_{cnt(j)}(b \to c) = \textsf{\textbf{false}}$.

Case 2: $(a \equiv a \mathcal{W} b)$. Let $b \mathcal{W} c \in pos(P_i)$, and as in previous subproofs, we know that there exists some $j \in [i, idx(cnt(i)))$ at which the $(\mathcal{W}^+)$ rule is applied to a in P_j. At this point, we know that either $c \in pos(P_{j+1})$, or $b \wedge \bullet b \mathcal{W} c \in pos(P_{j+1})$. In the first case, we know by the inductive hypothesis that $K^n_{cnt(j)}(c) = K^n_{cnt(i)}(c) = \textsf{true}$ and so we have that $K^n_{cnt(i)}(b \mathcal{W} c) = \textsf{true}$. Otherwise, $b \wedge \bullet b \mathcal{W} c \in pos(P_{j+1})$. So, by a similar argument, we know that $K^n_{cnt(i)}(b) = \textsf{true}$. We also know that $\bigcirc \neg b \mathcal{W} c \in neg(P_{idx(cnt(i))})$, and that $\neg b \mathcal{W} c \in neg(P_{idx(cnt(i))+1})$. So there is some index $k \in (idx(cnt(i)), idx(idx(cnt(i)+1))) \subset \mathbb{N}$, such that $a \in pos(P_k)$.

Continue with this process iteratively until the end of the path. One of two outcomes will occur. Either for every $k \geq cnt(i)$, we see that $K^n_k(b) = \textsf{true}$ or there exists some $k \geq cnt(i)$ such that $K^n_k(c) = \textsf{true}$ and for every $j \in (cnt(i), k)$, $K^n_j(b) = \textsf{true}$. Either way, we can conclude that $K^n(b \mathcal{W} c) = \textsf{true}$.

The proof is similar for $b \mathcal{W} c \in neg(P_i)$.

$\triangle$

Lemma 3.57 (Tableau Satisfiability)**.** *A tableau $\mathcal{T}_{\mathcal{P}}$ is a successful tableau for a PNP $\mathcal{P}$ if and only if $\widehat{\mathcal{P}}$ is satsifiable.*

Proof. We prove each direction separately:

($\Rightarrow$) Let $\mathcal{T}_{\mathcal{P}}$ be a tableau for a PNP $\mathcal{P}$. Assume that $\widehat{\mathcal{P}}$ is satisfiable, to show that $\mathcal{T}_{\mathcal{P}}$ is successful. Since $\widehat{\mathcal{P}}$ is satisfiable, there is some Kripke structure K^n such that $K_0(\widehat{\mathcal{P}}) = \textsf{true}$. Then, Lemma 3.53 gives that $\pi^{K^n}_{\mathcal{P}}$ in $\mathcal{T}_{\mathcal{P}}$ does not contain any closed tableau node, and hence, the root is not closed, and the tableau successful.

($\Leftarrow$) Let $\mathcal{P}$ be a PNP, and assume that $\mathcal{T}_{\mathcal{P}}$ is successful. Let $P_0, P_1, \cdots P_n$ be a terminating path in $\mathcal{T}_{\mathcal{P}}$, whose existence was proven by Lemma 3.54. Let K^n be a temporal structure such that for every $v \in \mathbf{V}$ and $i \in \mathbb{N}$,

$$v \in pos(P_{st(i)}) \Rightarrow \eta_i(v) = \textsf{true}$$
$$v \in neg(P_{st(i)}) \Rightarrow \eta_i(v) = \textsf{false}$$

Then Lemma 3.56 shows that

$$A \in pos(P_i) \Rightarrow K_{cnt(i)}(A) = \textsf{\textbf{true}}$$
$$A \in neg(P_i) \Rightarrow K_{cnt(i)}(A) = \textsf{\textbf{false}}$$

for all formulae A and $i \in \mathbb{N}$. This gives us that $K_{cnt(0)}(\widehat{P_0}) = \textsf{\textbf{true}}$, which is exactly $K_0(\widehat{\mathcal{P}}) = \textsf{\textbf{true}}$, and so $\widehat{\mathcal{P}}$ is satisfiable.

$\triangle$

Theorem 3.58 (Decidability of LTL$_f$). *For every formula a of LTL$_f$, the satisfiability of a is decidable.*

Proof. To test if a formula a of LTL$_f$ is satisfiable, we can create a PNP for it $P = (\{a\}, \emptyset)$. For an arbitrary K^n, note that $K^n(\widehat{\mathcal{P}}) = K^n(a)$. This means that $\widehat{\mathcal{P}}$ is satsifiable exactly when a is satisfiable. Lemma 3.57 says that the tableau $\mathcal{T}_{\mathcal{P}}$ is successful if and only if there exists a K^n such that $K^n_0(\widehat{\mathcal{P}}) = \textsf{\textbf{true}}$. Since we can easily (via depth-first search) test whether $\mathcal{T}_{\mathcal{P}}$ is successful, we know whether it is decidable. $\triangle$

Chapter 4

Related Work, Applications and Further Directions

Now that we have seen the standard presentations of LTL and LTL$_f$, we can look at some extensions of the logics. First, we will consider a translation of LTL$_f$ to LTL and consider which formulae are equivalent in the two settings [7]. Then, we will consider a coinductive proof of completeness for LTL$_f$ (ours is *inductive*) [29]. Then we will add past time temporal operators to LTL. This so-called "past-time LTL" has been shown to be sound, complete, and decidable [24]. Then we will consider the application of Temporal Logic that motivated this work in the first place, network programming [4], and conclude with some remarks about other temporal logics.

4.1 Insensitivity to Infiniteness

It is important to note that LTL and LTL$_f$ are in fact different logics [7], i.e. there are nontrivial formulae that are satisfiable in LTL and unsatisfiable in LTL$_f$. The formula

$$\Diamond a \wedge \Box(a \rightarrow \Diamond b) \wedge \Box(b \rightarrow \Diamond a) \wedge \Box(\neg a \vee \neg b)$$

is one such formula. So, when we are trying to decide the satisfiability of formulae in LTL$_f$, it is insufficient to simply translate the formula to LTL and evaluate it there. Nonetheless, the line between LTL and LTL$_f$ is blurred [7]. We can define a translation finite from LTL$_f$ terms to LTL terms that relies on the infinite replication of a variable *done* that denotes the end of time. We define it below.

Definition 4.1 (LTL$_f$-LTL Translation)**.** Given a formula $f \in \mathit{LTL}_f(\mathbf{V})$, translate f into LTL by introducting a fresh proposition *done*, such that $\mathit{done} \notin \mathbf{V}$. Further, require that $\Diamond\, \mathit{done}$ holds, that $\Box(\mathit{done} \rightarrow \bigcirc\, \mathit{done})$, and that all other propositions are false, i.e. that $\Box(\mathit{done} \rightarrow \bigwedge_{v_1 \in \mathbf{V}} \neg v_1)$. Define the function $\mathsf{finite} : \mathit{LTL}_f(\mathbf{V}) \rightarrow \mathit{LTL}(\mathbf{V} \cup \{\mathit{done}\})$ by

$$\begin{aligned}
\mathsf{finite}(v) &= v \\
\mathsf{finite}(\mathsf{end}) &= \bigcirc\, \mathit{done} \\
\mathsf{finite}(\neg a) &= \neg \mathsf{finite}(a) \\
\mathsf{finite}(a \rightarrow b) &= \mathsf{finite}(a) \rightarrow \mathsf{finite}(b) \\
\mathsf{finite}(\bigcirc\, a) &= \bigcirc(\mathsf{finite}(a) \wedge \neg \mathit{done}) \\
\mathsf{finite}(a\, \mathcal{U}\, b) &= \mathsf{finite}(a)\, \mathcal{U}\, (\mathsf{finite}(a) \wedge \neg \mathit{done})
\end{aligned}$$

Given a finite Kripke structure $K^n = (\eta_1, \ldots, \eta_n)$, let

$$\mathsf{finite}(K^n) = (\eta_1, \ldots, \eta_n, \eta_\perp, \eta_\perp, \ldots),$$

where $\eta_\perp(\mathit{done}) = \textbf{true}$ and for every other formula c, $\eta_\perp(c) = \textbf{false}$.

Now we define a formula a to be *insensitive to infiniteness* a is true in both the finite and infinite setting under this translation, i.e.

$$\vDash_{K^n} a \Leftrightarrow \vDash_{\mathsf{finite}(K^n)} a.$$

So, we can define a formula that must be true on K^n to determine if a given formula a is insensitive to infiniteness.

Theorem 4.2 (Insensitivity to Infiniteness)**.** *A formula $a \in \mathit{LTL}_f(\boldsymbol{V})$ is insensitive to infiniteness if and only if the following $\mathit{LTL}(\boldsymbol{V} \cup \{\mathit{done}\})$ formula is valid:*

$$(\Diamond\, \mathit{done} \wedge \Box(\mathit{done} \rightarrow \bigcirc\, \mathit{done}) \wedge \Box(\mathit{done} \rightarrow \bigwedge_{v \in \boldsymbol{V}} \neg v) \rightarrow (c \leftrightarrow \mathit{finite}(c)))$$

Proof. See [7] for proof △

Example 4.3. The formula $\Diamond\, \neg a$ is sensitive to infiniteness. We can observe this simply by noting that $\Diamond\, \neg a$ is satisfiable but not valid in LTL$_f$. However, it is valid in our translated LTL$_f$. For any finite Kripke structure and any i, we see that $\mathsf{finite}(K^n)_i(\Diamond\, \neg a) = \textbf{true}$, because we know $\Diamond\, \mathit{done}$ evaluates to true, and that $\mathit{done} \rightarrow \neg a$ evaluates to true.

Remark 4.4. Example 4.3 points out a philisophical difference between LTL$_f$ and this translation into LTL. In LTL$_f$, once you reach the end of time, there is nothing, it doesn't make sense to ask if a formula is true or if it is false, because there is nothing to ask about. However, in these $\eta_\perp$ states, all variables are false, so things still exist, but hang around as ghosts.

A colloquial example highlighting this distinction regards a soccer match. After the conclusion of the 90 minutes, two fans might make very similar proclamations two versions of the same question, fan A says is "I hope Drogoba doesn't score in the next few minutes!", and fan B says "I hope Drogoba doesn't score again in the game!" Both of these allow the fans to conclude that Drogoba in fact won't score, but for fan A, it is because Drogoba is in the locker room and not scoring points, where for fan B it is because the game is over, and so of course, noone can score.

The authors of [7] are sure to stress that this is broadly insufficient for most practical purposes, and go on propose improved process modeling techniques for LTL$_f$ to make the logic more usable in model-checking. They provide a decision procedure based on the Past-Time Model 4.3, which runs in EXPTIME and PSPACE.

This work makes it clear that a distinct decision procedure for LTL$_f$ is necessary, as well as motivating the need for a sound and complete axiomatization, to allow for proofs about increasingly useful logic.

4.2 Coinductive Completeness

The first soundness and completeness results for LTL$_f$ used a coinductive axiomatic framework [29]. They use a completely analogous semantics, but use axioms and inference rules identified in Figure 4.1.

The main differences are the existence of the CoInduct and AlwStep rules. We have the same number of rules, but take a more natural approach, preferring to start at the beginning instead of at the end for our inductive inference rules.

Theorem 4.5 (Coinductive Soundness)**.** *The coinductive proof theory is sound.*

Proof. Show all of the axioms and inference rules are valid [29] △

$$\begin{array}{cr}
\text{all propositional tautologies} & (\textsc{Taut}) \\
\vdash \bullet(a \to b) \to (\bullet\, a \to \bullet\, b) & (\textsc{NextDistr}) \\
\vdash \neg \bullet\, a \to \bullet \neg a & (\textsc{NegNext}) \\
\vdash a\, \mathcal{W}\, b \leftrightarrow (b \vee a \wedge \bullet(a\, \mathcal{W}\, b)) & (\textsc{WkUntilUnroll}) \\
\dfrac{\vdash a}{\vdash \bullet\, a} & (\textsc{NextStep}) \\
\dfrac{\vdash \bullet\, a \to a}{\vdash a} & (\textsc{CoInduct}) \\
\dfrac{\vdash a}{\vdash \square\, a} & (\textsc{AlwStep})
\end{array}$$

Figure 4.1: Coinductive proof system for LTL_f

Theorem 4.6 (Coinductive Completeness)**.** *The coinductive proof theory is complete.*

Proof Idea. Create a graph of all possible PNPs and the transitions between them via a greatest fixpoint method. Prune away maximal connected components that do not contain satisfying paths [29]. △

4.3 An Extension to the Past

Past-Time LTL [24] introduces past-time operators into LTL. In this model of time, we still have a fixed starting point, and time only moves forwards, however we can now ask questions like "two states ago, did a always hold?". The syntax is presented below, with the previous state operator P, and the binary "back-to" operator B.

Definition 4.7 (Syntax for Past-Time *LTL* [24])**.** Let $\mathbf{V}$ be a set of variables, then we can define *the syntax for past-time LTL* as

$$\begin{array}{rll}
a, b \ ::= & v \in \mathbf{V} \ \mid\ \bot \ \mid\ a \to b \ \mid & \text{(From Classical)} \\
& \mathsf{X}\, a \ \mid\ a\, \mathsf{W}\, b \ \mid & \text{(From LTL)} \\
& \mathsf{P}\, a \ \mid\ a\, \mathsf{B}\, b & \text{(Past-Time operators)}
\end{array}$$

Note that we are using X to refer to the "next" operator, that we denoted as $\bigcirc$ in the previous section, and P to refer to the "last" operator, the past time version of X. We have similar changes for the operator meaning "always in the past", denoted as A, or "ever in the past", denoted as E. We maintain the pieces of syntactic sugar that we defined in Example 3.4, and adding several others:

Definition 4.8 (Syntactic Sugar for Past-Time LTL)**.** Define the following sugarings for past-time LTL

$$\mathsf{A}\, a \triangleq a \;\mathsf{B}\; \bot \tag{4.1}$$

$$\mathsf{E}\, a \triangleq \neg\, \mathsf{A} \neg a \tag{4.2}$$

$$a \;\mathsf{S}\; b \triangleq a \;\mathsf{B}\; b \wedge \mathsf{E}\, b \tag{4.3}$$

$$\mathsf{WP}\, a \triangleq \neg\, \mathsf{P} \neg a \tag{4.4}$$

$$\mathsf{WX}\, a \triangleq \neg\, \mathsf{X} \neg a \tag{4.5}$$

$$\mathsf{start} \triangleq \neg\, \mathsf{P}\, \top \tag{4.6}$$

The naming convention for the binary temporal operator is slightly different in the past setting. We pronounce B as "back to", referencing the past equivalent of W, whereas we pronounce S as "since", referencing the past equivalent of U. Accordingly, B is sometimes pronounced as "weak since."

Now that we have the syntax for the logic, we can define a similar $\models$ relation to the one developed in Section 3.2. The semantics are defined generally, for finite *or* infinite time. We will use the notation $K^{(n)}$ to refer to a kripke structure that could be finite or infinite, and K^n or K when we want to differentiate between the two cases.

Definition 4.9 (Valuation Function for Past-Time LTL)**.** Let $K^{(n)}$ be a (finite) Kripke structure, define the function $K_i^{(n)}$ forall $0 < i (< n)$ as

$$
\begin{aligned}
K_i^{(n)}(v) &= \eta_i(v) \\
K_i^{(n)}(\bot) &= \textsf{\textbf{false}} \\
K_i^{(n)}(a \to b) &= \begin{cases} \textsf{\textbf{true}} & \text{if } K_i^{(n)}(a) = \textsf{\textbf{false}} \\ \textsf{\textbf{true}} & \text{if } K_i^{(n)}(b) = \textsf{\textbf{true}} \\ \textsf{\textbf{false}} & \text{otherwise} \end{cases} \\
K_i^{(n)}(\mathsf{X}\, a) &= \begin{cases} K_{i+1}^{n}(a) & \text{if } i < n \\ \textsf{\textbf{false}} & \text{otherwise} \end{cases} \\
K_i^{(n)}(a \mathbin{\mathsf{W}} b) &= \begin{cases} \textsf{\textbf{true}} & \text{if } K_j^{(n)}(a) = \textsf{\textbf{true}}, \text{ for all } i \le j \le n \\ \textsf{\textbf{true}} & \text{if there exists } i \le k \le n, \text{ such that } K_k^{(n)}(b) = \textsf{\textbf{true}} \\ & \quad \text{and for every } j \text{ such that } i \le j < k, K_i^{(n)}(a) = \textsf{\textbf{true}} \\ \textsf{\textbf{false}} & \text{otherwise} \end{cases} \\
K_i^{(n)}(\mathsf{P}\, a) &= \begin{cases} K_{i-1}^{(n)}(a) & \text{if } i > 0 \\ \bot & \text{otherwise} \end{cases} \\
K_i^{(n)}(a \mathbin{\mathsf{B}} b) &= \begin{cases} \textsf{\textbf{true}} & \text{if } K_j^{(n)}(a) = \textsf{\textbf{true}}, \text{ for all } i \ge j \ge 0 \\ \textsf{\textbf{true}} & \text{if there exists } i \ge k \ge 0, \text{ such that } K_k^{(n)}(b) = \textsf{\textbf{true}} \\ & \quad \text{and for every } j \text{ such that } i \ge j > k, K_i^{(n)}(a) = \textsf{\textbf{true}} \\ \textsf{\textbf{false}} & \text{otherwise} \end{cases}
\end{aligned}
$$

These semantics are unsurprising, the forward-time operators have the exact same interpretaton, and the past-time operators begin interpreted in a completely analogous way. The last piece we need is the proof theory.

Definition 4.10 (Proof System for Past-Time LTL)**.** The axioms for Past-Time LTL are:

P1	$\vdash \neg\,\mathsf{P}\,a \leftrightarrow \mathsf{WP}\,\neg a$	F1	$\vdash \neg\,\mathsf{X}\,a \leftrightarrow \mathsf{WX}\,\neg a$
P2	$\vdash \mathsf{P}\,a \rightarrow \mathsf{WP}\,a$	F2	$\vdash \mathsf{X}\,a \rightarrow \mathsf{WX}\,a$
P3	$\vdash a \rightarrow \mathsf{WP}\,\mathsf{X}\,a$	F3	$\vdash a \rightarrow \mathsf{WX}\,\mathsf{P}\,a$
P4	$\vdash \mathsf{WP}(a \rightarrow b) \rightarrow (\mathsf{WP}\,a \rightarrow \mathsf{WP}\,b)$	F4	$\vdash \mathsf{WX}(a \rightarrow b) \rightarrow (\mathsf{WP}\,a \rightarrow \mathsf{WP}\,b)$
P5	$\vdash \neg\,\mathsf{E}\,a \leftrightarrow \mathsf{A}\,\neg a$	F5	$\vdash \neg\,\mathsf{F}\,a \leftrightarrow \mathsf{G}\,\neg a$
P6	$\vdash \mathsf{A}(a \rightarrow b) \leftrightarrow (\mathsf{A}\,a \rightarrow \mathsf{A}\,b)$	F6	$\vdash \mathsf{G}(a \rightarrow b) \leftrightarrow (\mathsf{G}\,a \rightarrow \mathsf{G}\,b)$
P7	$\vdash \mathsf{A}\,a \rightarrow \mathsf{WP}\,a$	F7	$\vdash \mathsf{G}\,a \rightarrow \mathsf{WX}\,a$
P8	$\vdash \mathsf{A}(a \rightarrow \mathsf{WP}\,a) \rightarrow (a \rightarrow \mathsf{A}\,a)$	F8	$\vdash \mathsf{G}(a \rightarrow \mathsf{WX}\,a) \rightarrow (a \rightarrow \mathsf{G}\,a)$
P9	$\vdash a\,\mathsf{S}\,b \leftrightarrow b \vee [a \wedge \mathsf{P}(a\,\mathsf{S}\,b)]$	F9	$\vdash a\,\mathsf{U}\,b \leftrightarrow b \vee [a \wedge \mathsf{X}(a\,\mathsf{U}\,b)]$
P10	$\vdash \mathsf{E}\,\mathsf{start}$	F10	$\vdash a\,\mathsf{U}\,b \rightarrow \mathsf{F}\,b$

And then we can define the three inference rules:

R1	For a propositional tautology p, $\vdash p$.
R2	$a \rightarrow b, a \vdash b$
R3	$a \vdash \mathsf{G}\,a \wedge \mathsf{A}\,a$

We can point out a clear difference between LTL and LTL$_f$. One might thing that the past-time axioms P1-10 would completely specify LTL$_f$ and the forward-time axioms F1-10 would completely specify LTL. However, we notice that the axiom P3 cannot even be stated in LTL$_f$. This is a key axiom to use and leverage in certain proofs [25, 24]. Hence, our system, which omits this ability to transition between past and forward time, and forces a finite model of time presents a truly distinct sytem.

We see that Past-Time LTL is decidable, [24], and that completeness is a direct consequence of their decision procedure, which uses a *greatest* fixed point approach, as opposed to our *least* fixpoint approach.

Theorem 4.11 (Decidability). *Satisfiability (and validity) in Past-Time LTL are decidable.*

Theorem 4.12 (Completeness). *Past-Time LTL is complete.*

4.4 Network Applications

With the rise of the Internet and the increasing interconnectedness of the global computational architecture, there is an increasing need for verifiable methods of computing across multiple connected devices. Until quite recently [12], the main method of creating networks was to write hardware-specific code on each of the individual servers and switches and hope to coordinate certain ideal properties of the network. This is extremely arduous to make rigorous, and invariably, networks created this way are severely succeptible to attackers, who can exploit this lack of rigor.

NetKAT is the proposed solution to this problem [1], creates a unified top-level language with an abstract hardware-agnostic representation of a switch, allowing for formal, rigorous reasoning about network topologies, and forwarding strategies. This formal system is called NetKAT, short for **Net**work **K**leene **A**lgebra with **T**ests. The basis of the language is Kleene Algebra (KA) used to model regular expressions. This is extended to Kleene Algebra with Tests (KAT), adding adds boolean comparison to the language, which adds axioms enabling network programming, creating NetKAT.

The true innovation of NetKAT is that it is a formal system that easily models network topology, allows for the programming of specific forwarding policies, and allows for axiomatic proofs regarding those policies. A problem here is that these specifications become intractably large as the topology increases in size. The reason is this: as a network packet traverses the network, it accumulates a packet history, which is a list of vectors, the head of which is the current state of the packet (destination, addresses, type, etc). Unfortunately NetKAT policies can only ask questions about the current state, the head of this list. This is a less-than-efficient construction, since we are storing this packet history, but are unable to do anything with it.

Temporal NetKAT [4], an extension of NetKAT, leverages the packet history to increase expressivity. To do this, Temporal NetKAT adds LTL_f to standard NetKAT. We have been primarily examining its expression for forwards time; however, as we saw in Section 4.3, we can easily turn it around to express statements about the packet history. Importantly, we might ask questions such as "Has the packet ever been through a firewall?" or "Did the packet just come from some secure switch x?".

Temporal NetKAT does have succeed in proving soundness and decidability properties. It is formulated use an equational theory instead of an axiomatic framework. So their axioms are in the form $a \equiv b$. Notice that this is effectively the same as saying $\vdash a \leftrightarrow b$. They also prove a modified version of Completeness for their equational theory. We present the LTL_f portion of the axioms translated into our metatheoretical notation in Figure 4.2

Theorem 4.13 (Temporal NetKAT Partial completeness)**.** *For two arbitrary formulae of Temporal NetKAT, if* $[\mathsf{start}; a] = [\mathsf{start}; b]$*, then* $\mathsf{start}; a \equiv \mathsf{start}; b$

Now that we have a proof of completeness for LTL_f, we can derive a proof

$$\vdash \bigcirc a \wedge \bigcirc b \leftrightarrow \bigcirc(a \wedge b) \qquad (\textsc{ltl-last-dist-seq})$$
$$\vdash \bigcirc a \vee \bigcirc b \leftrightarrow \bigcirc(a \vee b) \qquad (\textsc{ltl-last-dist-plus})$$
$$\vdash \bullet \top \qquad (\textsc{ltl-wlast-one})$$
$$\vdash a\,\mathcal{U}\,b \leftrightarrow b \vee a \wedge \bigcirc(a\,\mathcal{U}\,b) \qquad (\textsc{ltl-since-unroll})$$
$$\vdash \neg(a\,\mathcal{U}\,b) \leftrightarrow (\neg b)\,\mathcal{W}\,(\neg a \wedge \neg b) \qquad (\textsc{ltl-not-since})$$
$$\frac{\vdash a \to \bullet\, a \wedge b}{\vdash a \to \Box\, b} \qquad (\textsc{ltl-induction})$$
$$\vdash \Box\, a \to \Diamond(\textsf{end} \wedge a) \qquad (\textsc{ltl-finite})$$

Figure 4.2: LTL$_f$ Rules and axioms in Temporal NetKAT

of completeness for Temporal NetKAT.

Theorem 4.14 (Temporal NetKAT Completeness)**.** *If* $[a] = [b]$ *then* $a \equiv b$.

Proof Idea. We have some formulae a and b, and we know that $[a] = [b]$. This is translated to $[a \leftrightarrow b] = [\textbf{true}]$. Now, the following table links the Temporal NetKAT Axioms with equivalent LTL$_f$ axioms.

LTL$_f$ Rule	Temporal NetKAT Axiom
NextAndDistr	ltl-last-dist-seq
NextOrDistr	ltl-last-dist-plus
WkNextTop	ltl-wlast-one
UntilUnroll	ltl-since-unroll
NotUntil	ltl-not-since
Induction'	ltl-induction
AlwaysFinite	ltl-finite

The above table shows the equivalent versions of all of the LTL$_f$ related Temporal NetKAT Axioms. We can also show that our LTL$_f$ axioms can be proved in Temporal NetKAT. Hence, LTL$_f$ and the Temporal NetKAT Axioms are equivalent. △

4.5 Further Directions

We have shown soundness, completeness, and decidability for a finite version of the infinite temporal logics. What if we look at other kinds of temporal logics? A close variant of LTL, known as Linear Dynamic Logic, or LDL, does away with the assumption that every successor state is unique. It has already been studied in the finite setting [8], and it could be fairly straightforward to extend some of the observations and techniques made here to find soundness and completeness results for this logic. Computation tree logic, or CTL for short, is more complicated, in that it defines predicates in future temporal timelines, it is possible that soundness and completeness results could be found for some hypothetical CTL$_f$, or Finite Computation Tree Logic. We might also consider other temporal logics.

4.6 Conclusion

The main contribution of this work is a sound and complete proof theory for LTL$_f$ along with a decision procedure for satsifiability (Section 3). The proof structures and basic procedures largely outline the work presented by Kroger and Merz on Linear Temporal Logic [21], the general outline, graph construction and many of the definitions are largely based on their work. However, we have made several key additional contributions, aside from the translation to the finite setting. We formulated the system using the binary weak until ($\cdot\ \mathcal{W}\ \cdot$) operator instead of the always ($\Box\,\cdot$) operator, making our presentation slightly more general. We also have clarified many definitions and presented proofs in a slightly more formal way, with the addition of the comps function (Definition 3.29). Part of this is due to our presentation and some of it is due to our formalism, and part of it is due to the fact that the finite setting allows us to prove slightly simpler properties.

A further contribution is the insertion of the completeness and decidability results for LTL$_f$ into the Temporal NetKAT framework (Section 4.4). This extends their partail completeness and decidability results that hinge on packet histories having a symbolically identified start point, allowing the authors to derive general, compositional completeness and decidability results.

Our secondary contributions are a introduction to metamathematical inquiry intended for undergraduate mathematics students (Section 1), and a summary of the semantics, sound proof theory, and completeness properties for LTL (Section 2). This is a slightly modified version of what is presented by Kroger & Merz [21], again using the weak until operator instead of the always operator. We also present a summary of related results regarding LTL$_f$, in terms of related logics(Sections 4.2 and 4.3) and in terms of properties about

LTL$_f$ itself (Section 4.1).

We expect that these results will encourage the use of LTL$_f$ more application domains, and motivate further research into more efficient decision procedures.

Bibliography

[1] Anderson, C. J., Foster, N., Guha, A., Jeannin, J.-B., Kozen, D., Schlesinger, C., and Walker, D. Netkat: Semantic foundations for networks. In *ACM SIGPLAN Notices* (2014), vol. 49, ACM, pp. 113–126.

[2] Barringer, H., Goldberg, A., Havelund, K., and Sen, K. Program monitoring with ltl in eagle. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International* (2004), IEEE, p. 264.

[3] Barringer, H., Groce, A., Havelund, K., and Smith, M. Formal analysis of log files. *Journal of aerospace computing, information, and communication 7*, 11 (2010), 365–390.

[4] Beckett, R., Greenberg, M., and Walker, D. Temporal netkat. *ACM SIGPLAN Notices SIGPLAN Not. 51*, 6 (Feb 2016), 386401.

[5] Bornat, R. *Proof and disproof in formal logic: an introduction for programmers.* 2005.

[6] Cook, B., Koskinen, E., and Vardi, M. Temporal property verification as a program analysis task. In *International Conference on Computer Aided Verification* (2011), Springer, pp. 333–348.

[7] De Giacomo, G., De Masellis, R., and Montali, M. Reasoning on ltl on finite traces: Insensitivity to infiniteness. In *AAAI* (2014), Citeseer, pp. 1027–1033.

[8] De Giacomo, G., and Vardi, M. Y. Linear temporal logic and linear dynamic logic on finite traces. In *IJCAI'13 Proceedings of the Twenty-Third international joint conference on Artificial Intelligence* (2013), Association for Computing Machinery, pp. 854–860.

[9] De Giacomo, G., and Vardi, M. Y. Synthesis for ltl and ldl on finite traces. In *Proc. of IJCAI* (2015).

[10] De Morgan, A. *Formal logic: or, the calculus of inference, necessary and probable.* Taylor and Walton, 1847.

[11] Fix, L. Fifteen years of formal property verification in intel. In *25 Years of Model Checking*. Springer, 2008, pp. 139–144.

[12] Foster, N., Harrison, R., Freedman, M. J., Monsanto, C., Rexford, J., Story, A., and Walker, D. Frenetic: A network programming language. In *ACM Sigplan Notices* (2011), vol. 46, ACM, pp. 279–291.

[13] Fraer, R., Kamhi, G., Ziv, B., Vardi, M., and Fix, L. Efficient reachability computation both for verification and falsification. In *Proceedings of International Conference on Computer-Aided Design,(CAV00).*

[14] Giero, M. Weak completeness theorem for propositional linear time temporal logic. *Formalized Mathematics 20*, 3 (2012), 227–234.

[15] Gödel, K. Über formal unentscheidbare sätze der principia mathematica und verwandter systeme i. *Monatshefte für mathematik und physik 38*, 1 (1931), 173–198.

[16] Gödel, K. Some metamathematical results on completeness and consistency, on formally undecidable propositions of principia mathematica and related systems i, and on completeness and consistency. *From Frege to Gödel: A source book in mathematical logic 1931* (1962).

[17] Kern, C., and Greenstreet, M. R. Formal verification in hardware design: a survey. *ACM Transactions on Design Automation of Electronic Systems (TODAES) 4*, 2 (1999), 123–193.

[18] Kleene, S. C. *Introduction to metamathematics.* 1952.

[19] Kripke, S. Semantical considerations of the modal logic.

[20] Kropf, T. *Introduction to formal hardware verification.* Springer Science & Business Media, 2013.

[21] Kroger, F., and Merz, S. *Temporal logic and state systems.* Springer, 2008.

[22] Lamport, L. Time, clocks, and the ordering of events in a distributed system. *Communications of the ACM 21*, 7 (1978), 558–565.

[23] Lamport, L. "sometime" is sometimes "not never": On the temporal logic of programs. In *Proceedings of the 7th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (New York, NY, USA, 1980), POPL '80, ACM, pp. 174–185.

[24] Lichtenstein, O., Pnueli, A., and Zuck, L. The glory of the past. In *Workshop on Logic of Programs* (1985), Springer, pp. 196–218.

[25] Manna, Z., and Pnueli, A. The anchored version of the temporal framework. In *Workshop/School/Symposium of the REX Project (Research and Education in Concurrent Systems)* (1988), Springer, pp. 201–284.

[26] Nathanson, M. B. Desperately seeking mathematical truth. *The Best Writing on Mathematics 2010* (2011), 13.

[27] Nguyen, A. C., and Khoo, S. C. Towards automation of ltl verification for java pathfinder.

[28] Pnueli, A. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on* (Oct 1977), pp. 46–57.

[29] Roşu, G. Finite-trace linear temporal logic: Coinductive completeness. In *International Conference on Runtime Verification* (2016), Springer, pp. 333–350.

[30] Schubert, T. High level formal verification of next-generation microprocessors. In *Proceedings of the 40th annual Design Automation Conference* (2003), ACM, pp. 1–6.

[31] West, D. B., et al. *Introduction to Graph Theory*, vol. 2. Prentice hall Upper Saddle River, 2001.

Appendix A

Auxiliary Proofs

A.1 Semantics

Lemma A.1 (Totality). *For every Kripke structure K^n, every $i \in \{1, \ldots, n\}$, and an arbitrary formula a, then $K_i^n(a) = \textsf{true}$ or $K_i^n(a) = \textsf{false}$.*

Proof. Let $a \in LTL_f(\mathbf{V})$, and K^n be a finite Kripke structure of length n. Let $i \in \{1, \ldots, n\}$. It is easy to show by structural induction that $K_i^n(a)$ is defined. Hence, K_i^n is a total function on $LTL_f(\mathbf{V})$. $\triangle$

Lemma A.2 (Always Convergence). *Given a Kripke structure of length n, and a formula a of $LTL_f(\boldsymbol{V})$ then $K_n^n(\Box\, a) = K_n^n(a)$*

Proof. Immediate, by definition of the evaluation function. $K_n^n(\Box\, a) = \textsf{true}$ if and only if $K_j^n(a) = \textsf{true}$ for all $n \leq j \leq n$. So j must be n. Since $K_n^n(\Box\, a) = \textsf{true}$ if and only if $K_n^n(a) = \textsf{true}$, we can conclude that $K_n^n(\Box\, a) = K_n^n(a)$. $\triangle$

Corollary A.3 (Finite is Truncated). *Given a finite Kripke structure $K^n = (\eta_1, \eta_2, \cdots, \eta_n)$ and an infinite Kripke structure $K = (\eta'_1, \eta'_2, \eta'_3, \cdots)$, such that $\eta_i = \eta'_i$, $\vDash_{K^n} a$ if and only if $\vDash_{K[n]} a$.*

Proof. We proceed by induction on the size of K^n. For the base case, let $n = 1$. Then $K^1 = (\eta_1)$ and $K[1] = [\eta_1]$. So it is clear that $\vDash_{K^n} a$ if and only if $\vDash_{K[n]} a$.

Now consider a general n. Then $K^n = (\eta_1, \eta_2 \cdots, \eta_n)$, and $K[n] = [\eta_1, \eta_2, \cdots, \eta_n]$. Create modified structures $L^{n-1} = (\eta_2, \cdots, \eta_n)$ and $L = (\eta_2, \eta_3 \cdots, \eta_n, \cdots)$. Then by the induction hypothesis we know that $\vDash_{L^{n-1}} a$

if and only if $\vDash_{L[n-1]} a$. So for $K[n]_i(a) = K_i^n(a)$ for all a and $1 < i \leq n$. So we need to consider the case that $i = 1$. We see that $K[n]_1(a) = K_1^n(a)$ by definition and the inductive hypothesis. $\triangle$

Lemma A.4 (Semantic Modus Ponens)**.** *For a given set of formulae $\mathcal{F}$, with a, b formulae, if $\mathcal{F} \vDash a$ and $\mathcal{F} \vDash a \to b$, then $\mathcal{F} \vDash b$.*

Proof. Let $\mathcal{F}$ be a given set of formulae, and let a and b formula. Assume that (i) $\mathcal{F} \vDash a$ and (ii) $\mathcal{F} \vDash a \wedge b$ to show that $\mathcal{F} \vDash b$. Let K^n be a Kripke structure in which $\mathcal{F} \vDash_{K^n} a$, and $\mathcal{F} \vDash_{K^n} a \to b$. Let $i \in \{1, \ldots, n\}$, to show $K_i^n(b) = \textbf{true}$. By definition of $\vDash$, we have $K_i^n(a) = \textbf{true}$ (from (i)) and that either $K_i^n(a) = \textbf{false}$ or $K_i^n(b) = \textbf{true}$ (from (ii)). If we are in the case where $K_i^n(b) = \textbf{true}$, we are done; however the other case, wherein $K_i^n(a) = \textbf{false}$, is contradictory, since $K_i^n(a) = \textbf{true}$. So $K_i^n(b) = \textbf{true}$ and $\mathcal{F} \vDash b$.

$\triangle$

Lemma A.5 (Assumption of Temporal Operators)**.** *For a set of formulae $\mathcal{F}$, then $\mathcal{F} \cup \{a, \bigcirc \top\} \vDash \bigcirc a$ and $\mathcal{F} \cup \{a\} \vDash \Box a$.*

Proof. Let K^n be a finite Kripke structure such that $\vDash_{K^n} b$ for every $b \in \mathcal{F}$ and $i \in [n]$, including a. In otherwords, $\vDash_{K^n} a$ for every $i \in [n]$, which gives $\mathcal{F} \vDash \Box a$, and a specialization gives $K_{i+1}^n(a)$ for $i < n$, meaning $\mathcal{F} \vDash \bigcirc a$. We know that $i \neq n$ by contradiction. If it were the case, then $K(\bigcirc \top) = \textbf{false}$, which contradicts the assumption. $\triangle$

Corollary A.6 (Next Assumption)**.**

$$\bigcirc \top \vDash \bot$$

Proof. We want to show that for an arbitrary finite Kripke structure, K^n, such that $K_i^n(\bigcirc \top) = \textbf{true}$, we can derive $K_i^n(\bot) = \textbf{true}$. However, in fact, the assumption fails, because $K_n^n(\bigcirc \top) = \textbf{false}$. The result follows. $\triangle$

A.2 Proof Theory

Theorem A.7 (Soundness Theorem for LTL$_f$)**.** *Let a be a formula and $\mathcal{F}$ a set of formulae. If $\vdash a$, then $\vDash a$.*

Proof. By structural induction on $\vdash a$:

(Taut) a is a propositional tautology. Following Theorem 2.2.1 in Kroger-Merz [21], all propositional tautologies are valid. Their proof applies for our syntax, replacing formulae *mutatis mutandis.*

(NextDistr) We want to show that $\vDash \bullet(a \to b) \leftrightarrow (\bullet\, a \to \bullet\, b)$. Show each direction separately.

($\to$) Let K^n be an arbitrary finite Kripke Structure and i an arbitrary index, to show that $K^n_i(\bullet(a \to b) \to (\bullet\, a \to \bullet\, b) = \textsf{\textbf{true}}$. This breaks down into two cases.

($K^n_i(\bullet(a \to b)) = \textsf{\textbf{false}}$) Immediate.

($K^n_i(\bullet(a \to b)) = \textsf{\textbf{true}}$) Now, we must show that $K^n_i(\bullet\, a \to \bullet\, b) = \textsf{\textbf{true}}$. Now, if $i = n$, then $K^n_i(\bullet\, b) = \textsf{\textbf{true}}$ (because time has ended) and so $K^n_i(\bullet\, a \to \bullet\, b) = \textsf{\textbf{true}}$, and we're done. Otherwise, $i \neq n$, and so we can step the valuation function to see $K^n_i(\bigcirc \neg(a \to b) = \textsf{\textbf{false}}$, and again to see $K^n_{i+1}(\neg(a \to b) = \textsf{\textbf{false}})$ and once more to see that $K^n_{i+1}(a \to b) = \textsf{\textbf{true}})$. Now, we have two cases

($K^n_{i+1}(a) = \textsf{\textbf{false}}$) Now we can add the $\neg$ constructor and see that $K^n_{i+1}(\neg a) = \textsf{\textbf{true}}$ and again the $\bigcirc$ constructor to get $K^n_i(\bigcirc \neg a) = \textsf{\textbf{true}}$, and once more apply the $\neg$ rules, and add the $\bullet$ sugar to get $K^n_i(\bullet\, a) = \textsf{\textbf{false}}$. Finally we cann apply the $\to$ construction and get $K^n_i(\bullet\, a \to \bullet\, b)$.

($K^n_{i+1}(b) = \textsf{\textbf{true}}$) Add the $\neg$ constructor and its valuation to get $K^n_{i+1}(\neg b) = \textsf{\textbf{false}}$, followed by the same for $\bigcirc$, to see $K^n_{i+1}(\bigcirc \neg b) = \textsf{\textbf{false}}$. Then, once more negation and the syntactic sugar for weak next gives $K^n_i(\bullet\, b) = \textsf{\textbf{true}}$, which allows the conclusion $K^n_i(\bullet\, a \to \bullet\, b) = \textsf{\textbf{true}}$ using the implication valuation rule.

($\leftarrow$) Let K^n be an an arbitrary finite Kripke structure, and i be an arbitrary index, to show that $K^n_i((\bullet\, a \to \bullet\, b) \to \bullet(a \to b)) = \textsf{\textbf{true}}$. If $K^n_i(\bullet\, a \to \bullet\, b) = \textsf{\textbf{false}}$, then we're done. Otherwise, $K^n_i(\bullet\, a \to \bullet\, b) = \textsf{\textbf{true}}$ and we must show that $K^n_i(\bullet(a \to b)) = \textsf{\textbf{true}}$. Now, we have two cases:

($K^n_i(\bullet\, a) = \textsf{\textbf{false}}$) We can desugar $\bullet\, a$ to $\neg \bigcirc \neg a$. Then we can step the valuation function forward to get $K^n_{i+1}(a) = \textsf{\textbf{false}}$. Then we can see $K^n_{i+1}(b) = \textsf{\textbf{true}}$. Now using the implication constructor we can see $K^n_{i+1}(a \to b) = \textsf{true}$, which again using the weak next sugar lets us conclude $K^n_i(\bullet(a \to b) = \textsf{true}$.

($K^n_i(\bullet\, b) = \textsf{true}$) If $i = n$ we can immediately conclude $K^n_i(\bullet(a \to b)) = \textsf{true}$. Otherwise, We use the weak next sugar to $\bullet\, a$ to $\neg \bigcirc \neg a$, which gives $K^n_{i+1}(b) = \textsf{true}$. Then using the implication case for the valuation function , we can construct $K^n_{i+1}(a \to b) = \textsf{true}$, which again using the weak next sugar, lets us conclude that $K^n_i(\bullet\, a \to b) = T$.

(EndNextContra) We want to show that $\vDash \textsf{end} \to \neg \bigcirc a$. So given some finite Kripke structure K^n and an arbitrary index i, we have two cases, either $K^n_i(\textsf{end}) = \textsf{false}$ and we're done, or $K^n_i(\textsf{end}) = K^n_i(\neg \bigcirc \top) = \textsf{true}$ and we need to show that $K^n_i(\neg \bigcirc a) = \textsf{true}$. The only way that $K^n_i(\neg \bigcirc \top)$ could be **false** is if $i = n$, in which case $K^n_i(\bigcirc a)$ is also **false**. Then applying the negation case of the valuation function gives $K^n_i(\neg \bigcirc a) = \textsf{true}$.

(Finite) Assume we have a proof that $\vdash \Diamond \textsf{end}$. We want to show that $\vDash \Diamond \textsf{end}$. We can desugar this to a form that the K^n_i functions will understand, namely $\vDash \neg((\bigcirc \top)\ \mathcal{W}\ \bot)$. Take an arbitrary function Kripke structure K^n and an arbitrary index i, to show that $K^n_i(\neg((\bigcirc \top)\ \mathcal{W}\ \bot)) = \textsf{true}$. Equivalently, we show that $K^n_i(((\bigcirc \top)\ \mathcal{W}\ \bot)) = \textsf{false}$. Since we know that $K^n_k(\bot) = \textsf{false}$ for all possible k, we must find an index $j \in [i, n]$ such that $K^n_j(\bigcirc \top) = \textsf{false}$. Let $j = n$, then by definition $K^n_j(\bigcirc \top) = K^n_n(\bigcirc \top) = \textsf{false}$.

(WkUntilUnroll) We have derivation of $\vdash (a\ \mathcal{W}\ b) \to b \vee a \wedge \bullet(a\ \mathcal{W}\ b)$. We must show that $\vDash (a\ \mathcal{W}\ b) \to b \vee a \wedge \bullet(a\ \mathcal{W}\ b)$, specifically, we must show that for an arbitrary finite Kripke structure K^n and arbitrary index $i \in \{1, \ldots, n\}$, that $K^n_i((a\ \mathcal{W}\ b) \to b \vee a \wedge \bullet(a\ \mathcal{W}\ b)) = \textsf{true}$. By definition, we can equivalently show that $K^n_i(a\ \mathcal{W}\ b) = \textsf{false}$, or $K^n_i(b \vee a \wedge \bullet(a\ \mathcal{W}\ b)) = \textsf{true}$. Assume $K^n_i(a\ \mathcal{W}\ b) = \textsf{true}$ (since otherwise we are done), and show that $K^n_i(b \vee a \wedge \bullet(a\ \mathcal{W}\ b) = \textsf{true}$. So, we know that there exists some $j \in [i, n]$ such that $K^n_j(b) = \textsf{true}$ and for all $k \in [i, j), K^n_k(a) = \textsf{true}$. There are two cases, $j = i$, or $j > i$. If $j = i$, then we know that $K^n(b) = \textsf{true}$, and the result follows by definition. Otherwise, $j > i$, and so $K^n_i(a) = \textsf{true}$. Note also that either $K^n_{i+1}(a\ \mathcal{W}\ b) = \textsf{true}$, or $i = n$. If $i = n$ we are done, so consider the other case, which by using the *next* case of the valuation function definition, and then the weak next syntactic sugar to create $K_i^n(\bigcirc(a \mathcal{W} b) = K_i^n(\bullet(a \mathcal{W} b) = \textsf{true}$. Then, by the implication case and the disjunctive syntactic sugar we can construct $K_i^n(b \wedge a \vee \bullet(a \mathcal{W} b) = \textsf{true}$

(WkNextStep) We know that $a \vdash \bullet a$, so we need to show that $a \vDash \bullet a$. Consider an arbitrary K^n such that $K_i^n(a) = \textsf{true}$ for all $i = 1, \ldots, n$. We need to show that for an arbitrary K_i^n, $K_i^n(\bullet a) = \textsf{true}$. There are two cases, either $i = n$, or $i < n$. If $i = n$, then we need to show that $K_n^n(\bigcirc \neg a) = \textsf{false}$, which holds by definition. Otherwise, show that $K_i^n(\bigcirc \neg a) = K_{i+1}^n(\neg a) = \textsf{false}$. Equivalently show that $K_{i+1}^n(a) = \textsf{true}$, which we know from our construction of K^n.

(Induction) We assume that we have $\vdash b \to c$ and $\vdash b \to \bullet b$. The induction hypothesis allows us to interpret these semantically, i.e. $\mathcal{F} \vDash b \to c$ and $\mathcal{F} \vDash b \to \bullet b$. Let K^n be an arbitrary Kripke structure such that $\mathcal{F} \vDash_{K^n} (b \to c)$ and $\mathcal{F} \vDash_{K^n} b \to \bullet b$. So if $K_i^n(b) = \textsf{false}$, we can use the implication case of the valuation function to show $K_i^n(b \to \Box c)$. So, assume that $K_i^n(b) = \textsf{true}$. Then, $K_i^n(c) = \textsf{true}$ and $K_i^n(\bullet b) = \textsf{true}$. We want to show $K_i^n(\Box b)$, i.e. for all $i \leq j \leq n$, $K_j^n(\Box b)$. When $i = j$ the result follows by assumption, and when $j = n$, the result follows by definition of the weak next case of the valuation function. In every other case (such that it exists), we can assume $K_{j-1}(\bullet a) = \textsf{true}$, and since $j < n$, we know $K_j^n(a) = \textsf{true}$ by the definition of the weak next sugar, and the negation and next cases of the valuation function.

△

Theorem A.8 (Deduction Theorem)**.** *Let a, b be formulae, $\mathcal{F}$ a set of formulae. $\mathcal{F}, a \vdash b$ if and only if $\mathcal{F} \vdash \Box a \to b$.*

Proof. Prove each direction separately.

($\Rightarrow$) We assume that $\mathcal{F}, a \vdash b$ to show that $\mathcal{F} \vdash \Box a \to b$. Proceed by structural induction on the derivation of b from $\mathcal{F}, a$.

Case 1: Assume that b is an axiom, or $b \in \mathcal{F}$. Then, $\mathcal{F} \vdash b$ and $\mathcal{F} \vdash \Box a \to b$ follows with Taut

Case 2: Assume that $b \equiv a$. Then we know from AlwaysUnroll, that $\vdash \Box a \to (a \wedge \bullet \Box a)$, which simplifies to $\vdash \Box a \to a$ by Taut. Now, we can freely add more conditions and see that $\mathcal{F} \vdash \Box a \to a$.

Case 3: Assume that $b \equiv \bullet c$ is a conclusion of WkNextStep. So, we must also have $\mathcal{F}, a \vdash c$. Then applying the induction hypothesis, we get $\mathcal{F} \vdash \Box a \to c$. Now we show that from these we can derive $\Box a \to \bigcirc c$.

1.	$\Box a \to c$	shown so far
2.	$\Box a \to \bigcirc \top$	shown so far
3.	$\bullet(\Box a \to c)$	WkNext, 1
4.	$\bullet \Box a \to \bullet c$	WkNextDistr, 3
5.	$\Box a \to a \wedge \bullet \Box a$	AlwUnroll
6.	$\Box a \to \bullet \Box a$	Taut, 5
7.	$\Box a \to \bullet c$	Taut, 6, 4

Case 4: Assume that $b \equiv c \to \Box d$. Is a conclusion of Induction with premises $c \to d$ and $c \to \bullet c$. Then we get the inductive hypotheses that $\mathcal{F} \vdash \Box a \to (c \to d)$ and $\mathcal{F} \vdash \Box a \to (c \to \bullet c)$. Then, we can derive $\Box a \to (c \to \Box d)$ by Taut, WkNextAnd-Distr and Induction.

1.	$\Box a \to (c \to d)$	*derivable*
2.	$\Box a \to (c \to \bullet c)$	*derivable*
3.	$\Box a \wedge c \to d$	Taut, 1
4.	$\Box a \wedge c \to \bullet c$	Taut, 1
5.	$\Box a \to \bullet \Box a$	Taut, AlwUnroll
6.	$\Box a \wedge c \to \bullet \Box a \wedge \bullet c$	Taut, 4, 5
7.	$\bullet \Box a \wedge \bullet c \to \bullet(\Box a \wedge c)$	WkNextAndDistr
8.	$\Box a \wedge c \to \bullet(\Box a \wedge c)$	Taut, 6, 7
9.	$\Box a \wedge c \to \Box d$	Induction, 3, 8
10.	$\Box a \to c \to \Box d$	Taut, 9

($\Leftarrow$) Assume that $\mathcal{F} \vdash \Box a \to b$ to show that $\mathcal{F}, a \vdash b$. If $\mathcal{F} \vdash \Box a \to b$, then $\mathcal{F}, a \vdash \Box a \to b$. Then, since $\mathcal{F}, a \vdash a$, we need to show $\mathcal{F}, a \vdash \Box a$, or more specifically that $a \vdash \Box a$. Then we get $\mathcal{F}, a \vdash b$ by Taut.

It remains to be shown that $a \vdash \Box a$. Assume a. Then by WkNext we get $\bullet a$. Then by Taut we derive $a \to \bullet a$. Then by Induction, we get $a \to \Box a$, and then finally by Taut, we get $\Box a$.

$\triangle$

Lemma A.9 (Next-Step Implication). *Let $\mathcal{P}$ be a PNP*

$$\vdash \widehat{\mathcal{P}} \to \bullet \widehat{\sigma(\mathcal{P})}$$

Proof. By cases on σ_i, $\vdash \widehat{\mathcal{P}} \to \bullet c$ when $c \in \sigma_1(\mathcal{P}) \cup \sigma_2(\mathcal{P})$, and $\vdash \widehat{\mathcal{P}} \to \bullet \neg c$ when $c \in \sigma_3(\mathcal{P}) \cup \sigma_4(\mathcal{P})$. The proposition follows directly from this and WkNextAndDistr. In fact the cases will be for when $c \in \sigma_i^{+/-}(\mathcal{P})$, for each $i = 1, 2, 3, 4, 5$.

1. If $c \in \sigma_1^+(\mathcal{P})$, then $\bigcirc c \in pos(\mathcal{P})$, so $\widehat{\mathcal{P}} \to \bullet c$ by Taut and $\widehat{\mathcal{P}} \to \bullet c$.

2. If $c \in \sigma_2^+(\mathcal{P})$ then $c \equiv a \mathcal{W} b \in pos(\mathcal{P})$, and $b \in neg(\mathcal{P})$. So, $\vdash \widehat{\mathcal{P}} \to \neg b \wedge a \mathcal{W} b$ by Taut. Then, WkUntilUnroll proves $\widehat{\mathcal{P}} \to \neg b \wedge (b \vee a \vee \bullet a \mathcal{W} b)$. Conclude that $\widehat{\mathcal{P}} \to \bullet a \mathcal{W} b$ holds by Taut.

3. If $c \in \sigma_3^+(\mathcal{P})$ then $c \equiv a \mathcal{W} b \in pos(\mathcal{P})$. We know that $\vdash \widehat{\mathcal{P}} \to \mathsf{end}$, so $\vdash \widehat{\mathcal{P}} \to \bullet \widehat{\sigma(\mathcal{P})}$ is apparent by CommNegNext.

4. If $c \in \sigma_4^-(\mathcal{P})$, then $\bigcirc c \in neg(\mathcal{P})$. We can say that $\vdash \widehat{\mathcal{P}} \to \neg \bigcirc c$. We want to show that $\widehat{\mathcal{P}}$ implies $\bullet \neg c$, which is equivalent to $\neg \bigcirc \neg\neg c$, which is then equivalent to $\neg \bigcirc c$, by Taut. Hence $\vdash \widehat{\mathcal{P}} \to \bullet \neg c$.

5. If $c \in \sigma_5^-(\mathcal{P})$, then $\Box c \in neg(\mathcal{P})$ and $c \in pos(\mathcal{P})$, so $\vdash \widehat{\mathcal{P}} \to (\neg \Box c \wedge c)$, then by AlwUnroll, we get $\vdash \widehat{\mathcal{P}} \to c \wedge (\neg c \vee \bigcirc \neg \Box a)$. Then by Taut, we can simplify this to $\vdash \bigcirc \Box c$, and by NextWkNext, conclude $\vdash \widehat{\mathcal{P}} \to \bullet \neg \Box c$.

$\triangle$

A.3 Completeness

Lemma A.10 (PNPs are Well-Behaved). *Let $\mathcal{P} = (\mathcal{F}^+, \mathcal{F}^-)$ be a consistent PNP, and a, b be formulae, then*

1. $\mathcal{F}^+$ and $\mathcal{F}^-$ are disjoint

2. Either $(\mathcal{F}^+, \mathcal{F}^- \cup \{a\})$ or $(\mathcal{F}^+ \cup \{a\}, \mathcal{F}^-)$ is a consistent PNP

3. $\bot \notin \mathcal{F}^+$

4. If, $a, b, a \to b \in \mathcal{F}_\mathcal{P}$, then if $a \in \mathcal{F}^-$ or $b \in \mathcal{F}^+$, $a \to b \in \mathcal{F}^+$, otherwise $a \to b \in \mathcal{F}^-$.

5. If $a \to b, a, b \in \mathcal{F}_\mathcal{P}$, then if $a \to b \in \mathcal{F}^+$, and $a \in \mathcal{F}^+$, then $b \in \mathcal{F}^+$.

Proof.

1. Assume $\mathcal{F}^+ \cap \mathcal{F}^-$ is nonempty, and pick some $a \in \mathcal{F}^+ \cap \mathcal{F}^-$. The $\vdash \widehat{\mathcal{P}} \to a \wedge \neg a$, so conclude $\vdash \neg\widehat{\mathcal{P}}$, which contradicts the consistency of $\vdash \widehat{\mathcal{P}}$.

2. If $a \in \mathcal{F}^+$ or $a \in \mathcal{F}^-$ we have $(\mathcal{F}^+ \cup \{a\}, \mathcal{F}^-) = \mathcal{P}$, or $(\mathcal{F}^+, \mathcal{F}^- \cup \{a\}) = \mathcal{P}$ and the assertion follows by the consistency of $\mathcal{P}$. Otherwise, we assume for the sake of contradiction, that both pairs under consideration are inconsistent, which in turn gives $\vdash \neg(\widehat{\mathcal{P}} \wedge a)$ and $\vdash \neg(\widehat{\mathcal{P}} \wedge \neg a)$. This gives a contradiction since Taut gives $\vdash \neg\widehat{\mathcal{P}}$. Hence at least one of the pairs must be consistent.

3. Assume $\bot \in \mathcal{F}^+$. Then $\vdash \widehat{\mathcal{P}} \to \bot$ by Taut, which contradicts the consistency of $\mathcal{P}$, so $\bot \notin \mathcal{F}^+$.

4. Assume that $a \to b \in \mathcal{F}^+$, but $a \notin \mathcal{F}^-$ and $b \notin \mathcal{F}^+$. Since $a, b \in \mathcal{F}_\mathcal{P}$, $a \in \mathcal{F}^+$ and $b \in \mathcal{F}^-$. Then $\vdash \widehat{\mathcal{P}} \to a \wedge \neg b \wedge (a \to b)$ which yield $\vdash \neg\widehat{\mathcal{P}}$ by Taut. The contradiction demonstrates that $a \in \mathcal{F}^-$ or $a \in \mathcal{F}^+$. To show the other direction, assume that $a \in \mathcal{F}^-$ or $b \in \mathcal{F}^+$. For the sake of contradiction assume that $a \to b \notin \mathcal{F}^+$ which means that $a \to b \in \mathcal{F}^-$. This gives $\vdash \widehat{\mathcal{P}} \to \neg(a \to b) \wedge \neg a$ or $\vdash \widehat{\mathcal{P}} \to \neg(a \to b) \wedge b$ which both give $\vdash \neg\widehat{\mathcal{P}}$ by Taut. Conclude $a \to b \in \mathcal{F}^+$.

5. Assume that $b \notin \mathcal{F}^+$. Then $b \in \mathcal{F}^-$, which gives $\vdash \widehat{\mathcal{P}} \to a \wedge \neg b$. We also see that $\vdash \widehat{\mathcal{P}} \to (a \to b)$. Then by Taut we get $\vdash \neg\widehat{\mathcal{P}}$, which proves that $b \in \mathcal{F}^+$ by contradiction to the consistency of $\mathcal{P}$.

$\triangle$

Lemma A.11 (Consistent Completion Existence)**.** *For $\mathcal{P}$, a consistent PNP, and*

$$\vdash \widehat{\mathcal{P}} \to \bigvee_{Q \in \mathsf{comps}(\mathcal{P})} \widehat{\mathcal{Q}}$$

Proof. Let $\mathcal{P}$ be a consistent PNP, and let $\mathcal{P}_1, \cdots, \mathcal{P}_m$ be those PNPs with disjoint positive and negative sets such that $\tau(\mathcal{P}) = \tau(\mathcal{P}_i)$, for all $i = 1, \ldots, m$. Let $\mathcal{P}'_1, \cdots, \mathcal{P}'_n$ be those $\mathcal{P}'_i$ which are in $\mathsf{comps}(\mathcal{P})$. Without loss of generality, relabel $\mathcal{P}_1, \cdots, \mathcal{P}_m$ such that $\mathcal{P}'_j = \mathcal{P}_i$ with $i = 1, \ldots, m$ and $j = 1, \ldots, n$.

Then, for $m > i > n$, either $\mathcal{P}_i$ is inconsistent, or $pos(\mathcal{P}) \nsubseteq pos(\mathcal{P}_i)$ and $neg(\mathcal{P}) \nsubseteq pos(\mathcal{P}_i)$. In either case, we obtain $\vdash \widehat{\mathcal{P}} \to \neg\widehat{\mathcal{P}'_i}$ by TAUT for every $i > n$. From Lemma 3.31, we get that $\vdash \bigvee_{i=1}^{m} \widehat{\mathcal{P}'_i}$. Then we can eliminate the bad cases, and get $\vdash \widehat{\mathcal{P}} \to \bigvee_{i=1}^{n} \widehat{\mathcal{P}'_i}$. △

Lemma A.12 (All Nodes Have Successors)**.** *Let $\mathcal{P}$ be a consistent and complete PNP, and $\mathcal{Q}_1, \cdots, \mathcal{Q}_n$ the nodes of $\mathcal{G}_{\mathcal{P}}$.*

$$\vdash \bigvee_{i=1}^{n} \widehat{\mathcal{Q}_i} \to \bullet \bigvee_{i=1}^{n} \widehat{\mathcal{Q}}_i$$

Proof. From Lemma 3.25, we have $\vdash \mathcal{Q}_i \to \bullet\widehat{\sigma(\mathcal{Q}_i)}$ for each $i = 1, \cdots n$. Let $\mathcal{Q}'_{i_1}, \cdots, \mathcal{Q}'_{i_m}$ be the m completions of $\sigma(\mathcal{Q}_i)$, by Lemma 3.33, we get that $\vdash \widehat{\sigma(\mathcal{Q}_i)} \to \bigvee_{j=1}^{n} \mathcal{Q}'_{i_j}$. Since by the definition of $\mathcal{G}_{\mathcal{P}}$ we have $\mathcal{Q}'_{i_j} \in \{\mathcal{Q}_i, \cdots, \mathcal{Q}_m\}$, which in our proof system is represented by $\vdash \widehat{Q'_{i_j}} \to \bigvee_{k=1}^{m} Q_k$ for each $j \in [m]$. Since each $\mathcal{Q}'_{i_j}$, (a completion of $\sigma(\mathcal{Q}_i)$) implies another node in the tree, and $\sigma(\mathcal{Q}_i)$ implies each of its completions, we can chain these to get $\vdash \widehat{\sigma(\mathcal{Q}_i)} \to \bigvee_{k=1}^{n} \widehat{\mathcal{Q}_k}$. Then, by WKNEXT and NEXTDISTR, we get $\vdash \bullet\widehat{\sigma\mathcal{Q}_i} \to \bullet\bigvee_{k+1} \widehat{\mathcal{Q}_k}$, and we get the result $\vdash \widehat{\mathcal{Q}_i} \to \bullet\bigvee_{k=1}^{n} \mathcal{Q}_k$ for each $i \in [n]$. The result follows by TAUT. △

Lemma A.13. *Every terminal path is a fulfilling path.*

Proof. Consider a terminal path $\pi = \mathcal{P}_1, \mathcal{P}_2, \cdots, \mathcal{P}_n, \mathcal{Z}$. Let S be the set of temporal formulae in π. Consider the terminal node $\mathcal{Z}$. Either there are no temporal formulae in $\mathcal{F}_{\mathcal{Z}}$, or there is at least one. In the first case, we are done, since every temporal formula has been fulfilled. So, consider the second case, where we have some set $S' = \mathcal{F}_{\mathcal{Z}} \cap S$ of exactly the temporal operators that haven't been fulfilled by the last state. We will show that they are fulfilled by the final state. The elements of S' can either be of the form $\bigcirc a$ or of the form $a \mathcal{W} b$, and can be either in the positive or negative sets. We will examine each of these cases:

Case 1 ($\bigcirc a \in \mathcal{F}_{\mathcal{Z}}$). This is trivial by definition of σ.

Case 2 ($a \mathcal{W} b \in \mathcal{F}_{\mathcal{Z}}$). Here we have two subcases:

Case 2a ($a \mathcal{W} b \in pos(\mathcal{Z})$). Since $\mathcal{Z}$ is consistent, we have $\vdash \widehat{\mathcal{Z}} \to b \vee a \wedge \bullet(a \mathcal{W} b)$ by WkUntilUnroll. Since we also have $\vdash \widehat{\mathcal{Z}} \to \textsf{end}$ by definition of a terminal node, we get $\vdash \widehat{\mathcal{Z}} \to b \vee a$. Since $b, a \in \tau(a \mathcal{W} b)$, we know that b or a is in $pos(\mathcal{Z})$, so $a \mathcal{W} b$ is fulfilled.

Case 2b ($a \mathcal{W} b \in neg(\mathcal{Z})$). We know $\vdash \widehat{\mathcal{Z}} \to \neg(a \mathcal{W} b)$, which gives $\vdash \widehat{\mathcal{Z}} \to \neg(b \vee a \wedge \bullet(a \mathcal{W} b)))$. Since we know $\vdash \widehat{\mathcal{Z}} \to \textsf{end}$, by the definition of a terminal node, $\bullet(a \mathcal{W} b)$ becomes $\top$. So we have $\vdash \widehat{\mathcal{Z}} \to \neg b \vee a$, or equivalently $\vdash \widehat{\mathcal{Z}} \to \neg b \wedge \neg a$. Since a and b are subformulae of $a \mathcal{W} b$, and $a \mathcal{W} b \in neg(\mathcal{Z})$, then also $a, b \in neg(\mathcal{Z})$. Hence $a \mathcal{W} b \notin \tau(\sigma(\mathcal{Z}))$.

$\triangle$

Theorem A.14 (Satisfiability Theorem for LTL$_f$)**.** *For any consistent PNP $\mathcal{P}$, the formula $\widehat{\mathcal{P}}$ is satisfiable.*

Proof. $\mathcal{P}' = (\{\Diamond \textsf{end}\} \cup pos(\mathcal{P}), neg(\mathcal{P}))$ is a consistent PNP, $\mathcal{P}^*$ is a completion of $\mathcal{P}'$. And $P_0, P_1, \cdots, P_n$ a complete, finite, rooted path of length n (by Lemma 3.41) in $\mathcal{G}_{\mathcal{P}*}$. Define $K^n = (\eta_0, \eta_1, \cdots, \eta_n)$ by

$$\begin{aligned} \eta_i(v) &= \textbf{true} && \text{if } v \in \mathcal{F}_i^+ \\ \eta_i(v) &= \textbf{false} && \textit{otherwise} \end{aligned}$$

We want to show that $K_0^n(\widehat{\mathcal{P}}) = \textbf{true}$. This is equivalent to showing that $K_0^n(\widehat{\mathcal{P}'}) = \textbf{true}$, which is also equivalent to showing that $K_0^n(\widehat{P_0}) = \textbf{true}$. Proving this is hard, so we will generalize the hypothesis making it easier to prove. We will show that for PNP P_i, and $f \in \mathcal{F}_{P_i}$, then $K_i^n(f) = \textbf{true}$ iff $f \in pos(P_i)$, and $K_i^n(f) = \textbf{false}$ otherwise.

We proceed by induction on the derivation of an arbitrary formula f:

$(f = v \in \mathbf{V})$. This is proved by the definition of K^n.

$(f = \bot)$. By definition of K_i^n, $K_i^n(\bot) = \textsf{false}$ and $\bot \notin \mathcal{F}_i^+$ by Lemma 3.22.

$(f = a \to b)$. Since $\mathcal{P}_i$ is a complete PNP, we have that $a, b \in \mathcal{F}^+ \cup \mathcal{F}^-$. The inductive hypothesis allows us to conclude that $K_i^n(a) = \textsf{false}$ iff $a \notin \mathcal{F}_i^+$. It also allows the conclusion $K_i^n(b) = \textsf{true}$ iff $b \in \mathcal{F}_i^+$. Then, $K_i^n(a \to b) = \textsf{true}$ if and only if $K_i^n(a) = \textsf{false}$ or $K_i^n(b) = \textsf{true}$, we can apply the inductive hypotheses to get $a \in \mathcal{F}_i^-$ or $b \in \mathcal{F}_i^+$. Then by Lemma 3.22, we can conclude that $a \to b \in \mathcal{F}_i^+$.

$(f = \bigcirc a)$. Lets break this down into a proof of each direction.

($\Rightarrow$) We want to show that if $K_i^n(\bigcirc a) = \textsf{true}$, then $\bigcirc a \in \mathcal{F}^+$. It must be that $i < n$, So by the definition of K_i^n, we know that $K_i^n(\bigcirc a) = \textsf{true}$ implies that $K_{i+1}^n(a) = \textsf{true}$. and so by the inductive hypothesis, we know that $a \in \mathcal{F}_{i+1}^+$. Then by Lemma 3.36, $\bigcirc a \in \mathcal{F}_i^+$.

($\Leftarrow$) Now lets show that if $\bigcirc a \in \mathcal{F}_i^+$, then $K_i^n(\bigcirc a) = \textsf{true}$. We know that since $\bigcirc a \in \mathcal{F}_i^+$, $\mathcal{P}_i$ is a consistent PNP, Lemma 3.36 means that $a \in \mathcal{F}_{i+1}^+$. Then by the induction hypothesis, $K_{i+1}^n(a) = \textsf{true}$, so $K_i^n(\bigcirc a)$, by definition of K^n.

$(f = a \mathcal{W} b)$. Here we dont need to break into cases, we can prove this directly. $K_i^n(a \mathcal{W} b) = \textsf{true}$ if and only if there exists $k \in [i, n]$ such that for all $j \in [i, k)$, $K_j^n(a) = \textsf{true}$ and $K_k^n(b) = \textsf{true}$. using the inductive hypothesis, we know this statement holds if and only if there exists $k \in [i, n]$ such that for all $j \in [i, k)$, $a \in pos(\mathcal{P}_j)$ and $b \in pos(\mathcal{P}_k)$. Then by Lemma 3.36 the previous statement holds if and only if $a \mathcal{W} b \in pos(\mathcal{P}_i)$.

$\triangle$

A.4 Decidability

Lemma A.15. *For every formula c and every finite Kripke structure K^n,*

$$K_n^n(c) = K_n^n(\mathsf{drop}(c))$$

Proof. Let c be an arbitrary formula, K^n an arbitrary Kripke structure. Show that $K^n_n(c) = K^n_n(c)$. By induction on the structure of c:

Case 1: $(c \equiv \bot)$. Since $\mathsf{drop}(\bot) = \bot$, and clearly $K^n_n(\bot) = K^n_n(\bot)$, the result follows.

Case 2: $(c \equiv v \in \mathbf{V})$. Since $\mathsf{drop}(v) = v$, the result follows.

Case 3: $(c \equiv a \to b)$. By definition, $\mathsf{drop}(a \to b) = \mathsf{drop}(a) \to \mathsf{drop}(b)$. Again, by definition $K^n_n(a \to b) = \textbf{true}$ if and only if $K^n_n(a) = \textbf{false}$ or $K^n_n(b) = \textbf{true}$. We can also expand the definition for $\mathsf{drop}(a) \to \mathsf{drop}(b)$ and see that $K^n_n(\mathsf{drop}(a) \to \mathsf{drop}(b)) = \textbf{true}$ if and only if $K^n_n(\mathsf{drop}(a)) = \textbf{false}$ or $K^n_n(\mathsf{drop}(b)) = \textbf{true}$. The inductive hypothesis gives that $K^n_n(\mathsf{drop}(a)) = K^n_n(a)$ and $K^n_n(\mathsf{drop}(b)) = K^n_n(b)$. So, We have that $K^n_n(\mathsf{drop}(a) \to \mathsf{drop}(b)) = \textbf{true}$ if and only if $K^n_n(a) = \textbf{false}$ or $K^n_n(b) = \textbf{true}$. Which then implies $K^n_n(\mathsf{drop}(a) \to \mathsf{drop}(b) = K^n_n(a \to b)$.

Case 4: $(c \equiv a \mathcal{W} b)$. We have that $\mathsf{drop}(a \mathcal{W} b) \equiv \mathsf{drop}(a) \vee \mathsf{drop}(b)$. Consider $K^n_n(a \mathcal{W} b)$; we know by Soundness and completeness that this is equivalent to $K^n_n(b \vee a \wedge \bullet(a \mathcal{W} b))$. We also know that $K^n_n(\bullet F) = \textbf{true}$ for any formula F. So we equivalently have to consider $K^n_n(b \vee a)$, which is $\textbf{true}$ when $K^n_n(b) = \textbf{true}$ or $K^n_n(a) = \textbf{true}$. The inductive hypothesis is that $K^n_n(b) = K^n_n(\mathsf{drop}(b))$ and $K^n_n(a) = K^n_n(\mathsf{drop}(a))$. This gives us that $K^n_n(b \vee a) = \textbf{true}$ if and only if $K^n_n(\mathsf{drop}(b)) = \textbf{true}$ or $K^n_n(\mathsf{drop}(a)) = \textbf{true}$. The by definition $K^n_n(b \vee a) = K^n_n(\mathsf{drop}(a) \vee \mathsf{drop}(b)) = K^n_n(\mathsf{drop}(a \mathcal{W} b)$.

△

Lemma A.16 (Rules are Sound)**.** *Let $\mathcal{T}_\mathcal{P}$ be a tableau for a PNP $\mathcal{P}$, and $\mathcal{Q}$ be some node of $\mathcal{T}_\mathcal{P}$. For all temporal structures K and all $i \in \mathbb{N}$:*

a) If $(\bigcirc)$ does not apply, then $K^n_i(\widehat{\mathcal{Q}}) = K^n_i(\widehat{\mathcal{Q}'})$ for some successor $\mathcal{Q}'$ of $\mathcal{Q}$.

b) If $(\bigcirc)$ applies, then $K^n_i(\widehat{\mathcal{Q}}) = K^n_{i+1}(\widehat{\mathcal{Q}'})$ for the successor $\mathcal{Q}'$ of $\mathcal{Q}$. Also, $\widehat{\mathcal{Q}'}$ is satisfiable only if $\widehat{\mathcal{Q}}$ is satsifiable.

Proof.

a) We will analyze each of the rules that could have applied, and show that the property holds for each of them:

   ($\perp$) If ($\perp$) applies to $\mathcal{Q}$, then $\mathcal{Q}$ has no successor, and the property holds vacuously.

   ($\rightarrow^+$) If ($\rightarrow^+$) applies to $\mathcal{Q}$ then there is some $a \rightarrow b \in pos(\mathcal{Q})$ that is being acted on. In the left successor, $\mathcal{Q}_l$, we have $b \in pos(\mathcal{Q}_l)$, and in the right successor, $\mathcal{Q}_r$, we have $a \in \neg(\mathcal{Q}_r)$. We know that $K_i^n(a \rightarrow b) =$ **true** if and only if $K_i^n(a) =$ **false** or $K_i^n(b) =$ **true**. Notice that $K_i^n(\widehat{\mathcal{Q}_l}) =$ **true** if and only if $K_i^n(a) =$ **false** and that $K_k^n(\widehat{\mathcal{Q}_r}) =$ **true**, if and only if $K_i^n(b) =$ **true**. And so we have that $K_i^n(\widehat{\mathcal{Q}}) =$ **true** if and only if $K_i^n(\widehat{\mathcal{Q}_r}) =$ **true** or $K_i^n(\widehat{\mathcal{Q}_r}) =$ **true**. The result follows.

   ($\rightarrow^-$) If ($\rightarrow^-$) applies to $\mathcal{Q}$ then there is some $a \rightarrow b \in pos(\mathcal{Q})$ that is being acted on. For the single successor $\mathcal{Q}'$, $a \in pos(\mathcal{Q}')$ and $b \in neg(\mathcal{Q}')$. Recall that $K^n(a \rightarrow b) =$ **false** if and only if both $K_i^n(a) =$ **true** and $K_i^n(b) =$ **false**. Since these are the only terms that change from $\mathcal{Q}$ to $\mathcal{Q}'$, we know that $K_i^n(\mathcal{Q}) = K_i^n(\mathcal{Q}')$.

   ($\mathcal{W}^+$) If ($\mathcal{W}^+$) applies to $\mathcal{Q}$, then there is some $a \ \mathcal{W} \ b \in pos(\mathcal{Q})$, and there are two successors, $\mathcal{Q}_l$ and $\mathcal{Q}_r$. We know that $b \in \mathcal{Q}_l$ and $a, \bullet(a \ \mathcal{W} \ b) \in pos(\mathcal{Q}_r)$. Soundness, Completeness, and BackUnroll give that $K_i^n(a \ \mathcal{W} \ b) =$ **true** if and only if $K_i^n(b) =$ **true** or $K_i^n(a \wedge \bullet(a \ \mathcal{W} \ b)) =$ **true**. This implies that $K_i^n(\widehat{\mathcal{Q}}) =$ **true** if and only if $K_i^n(\widehat{\mathcal{Q}_l}) =$ **true** or $K_i^n(\widehat{\mathcal{Q}_r}) =$ **true**. The result follows.

   ($\mathcal{W}^-$) If ($\mathcal{W}^-$) applies to $\mathcal{Q}$, then there is some $a \ \mathcal{W} \ b \in neg(\mathcal{Q})$. There are two succesors, $\mathcal{Q}_l$ and $\mathcal{Q}_r$. By definition, $a, b \in neg(\mathcal{Q}_l)$, $\bigcirc \neg(a \ \mathcal{W} \ b) \in pos(\mathcal{Q}_r)$. Then Soundness and Completeness give us that $K_i^n(a \ \mathcal{W} \ b) =$ **false**, if and only if $K_i^n(a \wedge b) =$ **true** or $K_i^n(\bigcirc \neg(a \ \mathcal{W} \ b)) =$ **false**. This gives us that $K_i^n(\widehat{\mathcal{Q}}) =$ **true** if and only if $K_i^n(\widehat{\mathcal{Q}_l}) =$ **true** or $K_i^n(\widehat{\mathcal{Q}_r}) =$ **true**. The result follows.

   (end) If (end) applies to $\mathcal{Q}$, then all of the elements of $\mathcal{F}_{\mathcal{Q}}$ are either the symbol $\perp$, a variable, or of the form $\bigcirc a$. Also $\bigcirc \top \in neg(\mathcal{Q})$ and $\sigma_1^+(\mathcal{Q}) = \emptyset$. Then $\mathcal{Q}' = (\mathsf{drop}(pos(\mathcal{Q})), \mathsf{drop}(neg(\mathcal{Q})) \cup \{\bigcirc \top\})$. There are no more positive temporal formulae, so we can consider $K_n^n(\widehat{\mathcal{Q}'})$. By Lemma 3.45, $K_{n-1}^n(\widehat{\mathcal{Q}}) = K_n^n(\widehat{\mathcal{Q}'})$.

b) Let $\mathcal{Q}$ be a node in $\mathcal{T}_{\mathcal{P}}$, and $\mathcal{Q}'$ a successor of $\mathcal{Q}$. Then we know that every formula in $pos(\mathcal{Q})$ and $neg(\mathcal{Q})$ is of the form $\bigcirc a$. We also know that $\bigcirc \bot \notin neg(\mathcal{Q})$. So for a given formula $\bigcirc a \in \mathcal{F}_{\mathcal{Q}}$, we know that $K_i^n(\bigcirc a) = K_{i+1}^n(a)$. Then we can conclude that $K_i^n(\mathcal{Q}) = K_{i+1}^n(\mathcal{Q}_i)$.

   Assume that $\widehat{\mathcal{Q}}$ is satisfiable. Let $K^n = (\eta_0, \eta_1, \dots, \eta_{n-1})$ be a satisfying model. We know that $K_0^n(\widehat{\mathcal{Q}}) = \textsf{true}$. We have just proven that $K_1^n(\widehat{\mathcal{Q}}) = \textsf{true}$. So if we let $K'^{n-1} = (\eta_1, \dots, \eta_{n-1})$, then it is clear that $K_0'^{n-1}(\widehat{\mathcal{Q}}) = \textsf{true}$.

$\triangle$

Lemma A.17 (Tableau Paths Not Closed). *Let $\mathcal{Q}$ be a node in a tableau and K^n a finite Kripke structure such that $K_0^n(\widehat{\mathcal{Q}}) = \textsf{true}$. Consider a path $\pi^{K^n}(\mathcal{Q}) = \mathcal{Q}_0, \mathcal{Q}_1, \dots, \mathcal{Q}_m$, then*

(a) For every $i = 0, \dots, m$, then $K_{cnt(i)}^n(\widehat{\mathcal{Q}_i}) = \textsf{true}$.

(b) $\pi^{K^n}(\mathcal{Q})$ does not contain any closed node, and

Proof. Let $\mathcal{Q}$ be an arbitrary node in a tableau, with K^n a Kripke structure such that K^n is a finite kripke structure with $K_0^n(\widehat{\mathcal{Q}}) = \textsf{true}$. Consider a path $\pi^{K^n}(\mathcal{Q}) = \mathcal{Q}_0, \mathcal{Q}_1, \dots, \mathcal{Q}_m$. Now show the desired properties

(a) Consider an arbitrary $i \in \{0, 1, \dots, m\}$, to show that $K_{cnt(i)}^n(\widehat{\mathcal{Q}_i}) = \textsf{true}$. We will do so inductively.

   First we let $i = 0$ and see that since $cnt(0) = 0$, $\mathcal{Q} = \mathcal{Q}_0$ and $K_0^n(\widehat{\mathcal{Q}_0}) = \textsf{true}$, we are done.

   Now consider the general case, where for every element $\mathcal{Q}_j$ of the path $\pi^K(\mathcal{Q})$ up to the current one, $\mathcal{Q}_i$,(i.e. $j < i$) has $K_{cnt(j)}^n(\widehat{\mathcal{Q}_j}) = \textsf{true}$. We are considering the next element, $\mathcal{Q}_i$, and we want to show that $K_{cnt(i)}^n(\widehat{\mathcal{Q}_i}) = \textsf{true}$. We have two cases. If $cnt(i-1) = cnt(i)$, then we know that $K_{cnt(i)}^n(\widehat{\mathcal{Q}_{i-1}}) = \textsf{true}$, and Lemma 3.47 gives $K_{cnt(i)}^n(\widehat{\mathcal{Q}_i}) = \textsf{true}$. Otherwise $cnt(i-1) + 1 = cnt(i)$, and $\mathcal{Q}$ is the only successor. Then by parts *b*) or *c*) of Lemma 3.47, we conclude that $K_{cnt(i)}^n(\widehat{\mathcal{Q}_i}) = \textsf{true}$.

(b) We want to show that $\pi^{K^n}(\mathcal{Q})$ has no closed node. Assume some $\mathcal{Q}_i$ in the path is closed according to one of the rules $(C1)$, $(C2)$, or $(C3)$. We proceed by induction on the $m = \pi^{K^n}(\mathcal{Q})$.

$(C1)$ If $(\perp)$ applies to a node $\mathcal{Q}_i$, $K^n(\widehat{\mathcal{Q}_i}) = \textsf{false}$, which contradicts part (a).

$(C2)$ Assume all the successors of $\mathcal{Q}_i$ are closed, but the induction hypothesis gives that at least one of them (specifically $\mathcal{Q}_{i+1}$) is not, hence $\mathcal{Q}_i$ is not closed.

$(C3)$ Assume that all terminating paths of $\mathcal{Q}_i$ have at least one closed node. Then for the successor of $\mathcal{Q}_i$ in $\pi^{K^n}(\mathcal{Q})$, namely $\mathcal{Q}_{i+1}$, we know that all terminating paths in the tableau, rooted at $\mathcal{Q}_{i+1}$, have at least one closed node. Hence, $\mathcal{Q}_{i+1}$ is closed. However, the induction hypothesis says that $\mathcal{Q}_{i+1}$ is not closed, and so there must exist a terminating path rooted at $\mathcal{Q}_i$ (in fact one such path is $\pi^k(\mathcal{Q})$).

△

Lemma A.18 (Path Modeling)**.** *Given a rooted, finite, terminating path $\pi = P_0, P_1, P_2, \ldots, P_n$, in a tableau $\mathcal{T}_\mathcal{P}$ for some PNP $\mathcal{P}$. If K^n is a temporal structure such that for every $v \in$ $\boldsymbol{V}$ and $i \in \mathbb{N}$,*

$$v \in pos(P_{idx(i)}) \Rightarrow \eta_i(v) = \textsf{true}$$
$$v \in neg(P_{idx(i)}) \Rightarrow \eta_i(v) = \textsf{false},$$

then for every $v \in$ $\boldsymbol{V}$ and $i \in \mathbb{N}$,

$$a \in pos(P_i) \Rightarrow K_{cnt(i)}(a) = \textsf{true}$$
$$a \in neg(P_i) \Rightarrow K_{cnt(i)}(a) = \textsf{false}.$$

Proof. Let $\mathcal{P}$ be a PNP, and $\mathcal{T}_\mathcal{P}$ be a tableau. Let $P_0, P_1, \ldots, P_n$ be a terminating path such that $P_0 = \mathcal{P}$. Let K^n be defined such that for every $i \in \{1, \ldots, n\}$ and variable $v \in \mathbf{V}$,

$$v \in pos(P_{idx(i)}) \Rightarrow \eta_i(v) = \textsf{true}$$
$$v \in neg(P_{idx(i)}) \Rightarrow \eta_i(v) = \textsf{false}.$$

Consider an arbitrary node P_i for some $i \in \{1, \ldots, n\}$, and a formula $a \in \mathcal{F}_{P_i}$. Show that $a \in pos(P_i)$ means that $K_{cnt(i)}(a) = \textsf{true}$ and $a \in neg(P_i)$ implies $K_{cnt(i)}(a) = \textsf{false}$. We will proceed by induction on the structure of the formula a:

Case 1: ($a \equiv v \in \mathbf{V}$). Assume $a \in pos(P_i)$, then $i \leq idx(cnt(i))$. In the construction of $\mathcal{T}_{\mathcal{P}}$ and the definitions of idx and cnt, note that no variables are removed until $idx(cnt(i)) + 1$. So it must be that $v \in pos(P_{idx(cnt(i))})$. Then the construction of K^n gives $K^n_{cnt(i)}(a) = \textsf{true}$.

Assume now that $a \in neg(P_i)$. Again by definition, $a \in neg(P_{idx(cnt(i))})$, and so $K^n_{cnt(i)}(a) = \textsf{true}$.

Case 2: ($a \equiv \bot$). If $\bot \in pos(P_i)$, we must be in a contradiction since we have assumed that P_i is not closed. So, the other option is that $\bot \in neg(P_i)$. By definition of any arbitrary valuation function, $K^n_j(\bot) = \textsf{false}$ for every $j \in \{1, \ldots, n\}$. So of course $K^n_{cnt(i)}(\bot) = \textsf{false}$.

Case 3: ($a \equiv b \to c$). Assume $a \in pos(P_i)$. At the current "time slice" we have the nodes $P_i, \ldots, P_{idx(cnt(i))}$. Since node $P_{idx(cnt(i))+1}$ represents an application of $(\textsf{end})$ or $(\bigcirc)$, we know that at some index $i \leq j < idx(cnt(i))$, the rule $(\to^+)$ is applied to a. By construction, it follows that $b \in neg(P_{j+1})$ or $c \in pos(P_{j+1})$. The induction hypothesis gives that $K^n_{cnt(j)}(A) = \textsf{true}$, and that $K^n_{cnt(j)}(A) = \textsf{true}$. This gives us $K^n_{cnt(j)}(A) = \textsf{true}$. Note that since $i \leq j < st(cnt(i))$, we know that $cnt(j) = cnt(i)$, and so we conclude that $K^n_{cnt(i)}(A) = \textsf{true}$.

Assume, contrarily, that $a \in neg(P_i)$. Consider the same "time slice" portion of the path as above, namely $P_i, \ldots, P_{idx(cnt(i))}$. By the same analysis there must be some index $i \leq j \leq idx(cnt(i))$ such that $(\to^-)$ is applied to a in P_j. This means that $b \in pos(P_{j+1})$ and $c \in neg(P_{j+1})$. The inductive hypothesis gives that $K^n_{cnt(j)}(a) = \textsf{true}$ and $K^n_{cnt(j)}(c) = \textsf{false}$. The definition of K^n and the fact that $cnt(i) = cnt(j)$ means that $K^n_{cnt(j)}(b \to c) = \textsf{false}$.

Case 4: ($a \equiv \bigcirc b$). Without loss of generality, let $\bigcirc b \in pos(P_i)$. We know that $\bigcirc b \in pos(P_{st(cnt(i))})$, by the construction of st, cnt, and $\mathcal{T}_{\mathcal{P}}$. Then, by tableau construction, $b \in \mathcal{F}_{P_{st(cnt(i))+1}}$. Note that $K^n_{cnt(i)}(b) = K^n_{cnt(i)+1}(b) = K^n_{idx(idx(cnt(i))+1)}(b)$. The inductive hypothesis gives that $K^n_{idx(idx(cnt(i))+1)}(b) = \textsf{true}$, and so conclude that $K^n_{idx(idx(cnt(i))+1)}(b)$

Case 5: $(a \equiv a\ \mathcal{W}\ b)$. Let $b\ \mathcal{W}\ c \in pos(P_i)$, and as in previous subproofs, we know that there exists some $j \in [i, idx(cnt(i)))$ at which the $(\mathcal{W}^+)$ rule is applied to a in P_j. At this point, we know that either $c \in pos(P_{j+1})$, or $b \wedge \bullet\, b\ \mathcal{W}\ c \in pos(P_{j+1})$. In the first case, we know by the inductive hypothesis that $K^n_{cnt(j)}(c) = K^n_{cnt(i)}(c) = \textsf{true}$ and so we have that $K^n_{cnt(i)}(b\ \mathcal{W}\ c) = \textsf{true}$. Otherwise, $b \wedge \bullet\, b\ \mathcal{W}\ c \in pos(P_{j+1})$. So, by a similar argument, we know that $K^n_{cnt(i)}(b) = \textsf{true}$. We also know that $\bigcirc \neg b\ \mathcal{W}\ c \in neg(P_{idx(cnt(i))})$, and that $\neg b\ \mathcal{W}\ c \in neg(P_{idx(cnt(i))+1})$. So there is some index $k \in (idx(cnt(i)), idx(idx(cnt(i)+1))) \subset \mathbb{N}$, such that $a \in pos(P_k)$.

Continue with this process iteratively until the end of the path. One of two outcomes will occur. Either for every $k \geq cnt(i)$, we see that $K^n_k(b) = \textsf{true}$ or there exists some $k \geq cnt(i)$ such that $K^n_k(c) = \textsf{true}$ and for every $j \in (cnt(i), k)$, $K^n_j(b) = \textsf{true}$. Either way, we can conclude that $K^n(b\ \mathcal{W}\ c) = \textsf{true}$.

The proof is similar for $b\ \mathcal{W}\ c \in neg(P_i)$.

△

A.5 Derived Rules in LTL$_f$

Lemma A.19. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \bigcirc a \to \neg\textsf{end} \qquad (\text{NextNotEnd})$$

Proof. We can start with the Taut $\vdash \textsf{end} \vee \neg\textsf{end}$. Then applying the contrapositive of EndNextContra to $\textsf{end}$ gives $\vdash \neg \bigcirc a \vee \neg\textsf{end}$, which desugars to $\vdash \bigcirc a \to \neg\textsf{end}$.

△

Lemma A.20. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \neg(\bigcirc \top \wedge \bigcirc \bot) \quad (\text{NextContraIsContra})$$

Proof. We know that $\vdash \top$, so WkNextStep gives $\vdash \bullet\, \top$. Then, by Taut, $\vdash \textsf{end} \vee \bullet\, \top$. We can desugar this to $\vdash \neg \bigcirc \top \vee \neg \bigcirc \neg \top$, which by DeMorgan's laws in Taut, gives $\vdash \neg(\bigcirc \top \wedge \bigcirc \bot)$.

$\triangle$

Lemma A.21. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \neg(\neg \bigcirc a \wedge \neg \bigcirc a \wedge \mathsf{end} \qquad (\textsc{CommNegNextR})$$

Proof. We see, Taut and EndNextContra give $\vdash \neg \bigcirc a \wedge \bigcirc \neg a \wedge \mathsf{end} \rightarrow \neg \bigcirc \wedge \bigcirc \neg a \wedge \neg \bigcirc \neg a \rightarrow \bot$. The result follows by Taut. $\triangle$

Lemma A.22. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \neg \bigcirc a \wedge \bigcirc \neg a \rightarrow \mathsf{end} \qquad (\textsc{CommNegNextR"})$$

Proof. We can write $\vdash \neg \bigcirc a \wedge \bigcirc \neg a \rightarrow (\mathsf{end} \vee \neg\mathsf{end}) \wedge \neg \bigcirc a \wedge \bigcirc \neg a$. Then we can distribute to get $\vdash \neg \bigcirc a \wedge \bigcirc \neg a \rightarrow (\mathsf{end} \wedge \neg \bigcirc a \wedge \bigcirc \neg a) \vee (\neg\mathsf{end} \wedge \neg \bigcirc a \wedge \bigcirc \neg a)$. Then by CommNegNextR' and Taut, we can conclude $\vdash \neg \bigcirc a \wedge \bigcirc \neg a \rightarrow (\neg\mathsf{end} \wedge \neg \bigcirc a \wedge \bigcirc \neg a)$ The result follows by Taut. $\triangle$

Lemma A.23. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \neg \bigcirc a \leftrightarrow \mathsf{end} \vee \bigcirc \neg a \qquad (\textsc{CommNegNext})$$

Proof. Consider each direction separately.

($\rightarrow$) We by Taut, we can see that $\vdash \neg \bigcirc a \rightarrow ((\neg \bigcirc a \wedge \bigcirc \neg a) \vee (\neg \bigcirc a \wedge \neg \bigcirc \neg a))$. Then by CommNegNextR" and Taut we see $\vdash \neg \bigcirc a \rightarrow (\mathsf{end} \vee \neg \bigcirc a \wedge \neg \bigcirc \neg a)$. Then Taut gives $\vdash \neg \bigcirc a \rightarrow (\mathsf{end} \vee \neg(\bigcirc \neg a \vee \bigcirc a))$, so by Taut we can write $\vdash \neg \bigcirc a \rightarrow \mathsf{end} \vee \bigcirc \neg a$.

($\leftarrow$) Equivalently show $\vdash (\neg\mathsf{end} \wedge \bullet\, a) \vee \neg \bigcirc a$.

Taut gives $\vdash (\mathsf{end} \vee \neg\mathsf{end}) \wedge (\bigcirc a \wedge \neg \bigcirc a)$, from which we can see

$$\vdash (\mathsf{end} \wedge \bigcirc a) \vee (\mathsf{end} \wedge \neg \bigcirc a) \vee (\neg\mathsf{end} \wedge \bigcirc a) \vee (\neg\mathsf{end} \wedge \neg \bigcirc a)$$

Note that $\vdash \neg\mathsf{end} \wedge \bigcirc a$ is a direct consequence of EndNextContra. So we now have

$$\vdash (\mathsf{end} \wedge \neg \bigcirc a) \vee (\neg\mathsf{end} \wedge \bigcirc a) \vee (\neg\mathsf{end} \wedge \neg \bigcirc a)$$

The forwards direction and TAUT gives

$$\vdash (\neg\mathsf{end} \land \bigcirc a) \lor (\neg \bigcirc a)$$

We now leverage the fact from TAUT that $\vdash \neg \bigcirc \neg a \lor \bigcirc \neg a$ to see

$$\vdash (\neg\mathsf{end} \land \bigcirc a \land \bigcirc \neg a) \lor (\mathsf{end} \land \bigcirc a \land \neg \bigcirc \neg a) \lor \neg \bigcirc a$$

Desugaring $\bigcirc a \land \bigcirc \neg a$ to $\neg(\neg \bigcirc a \lor \neg \bigcirc \neg a)$, we can then apply the forwards direction to see that

$$\vdash (\neg\mathsf{end} \land \neg(\neg\mathsf{end} \lor \bigcirc \neg a \lor \neg \bigcirc \neg a)) \lor (\mathsf{end} \land \bigcirc a \land \neg \bigcirc \neg a) \lor \neg \bigcirc a$$

The leftmost disjunction is contradictory by TAUT, so we can prove

$$\vdash (\mathsf{end} \land \bigcirc a \land \neg \bigcirc \neg a) \lor \neg \bigcirc a$$

△

Lemma A.24. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash (\bigcirc a \to \bigcirc b) \leftrightarrow \bigcirc(a \to b) \lor \mathsf{end} \qquad \text{(NEXTDISTR)}$$

Proof. First prove the following

$$\vdash \bullet a \land \neg\mathsf{end} \to \bigcirc a \qquad \text{(NEXTDISTRLEM)}$$

We can see from TAUT and COMMNEGNEXT that $\vdash \bullet a \land \neg\mathsf{end} \to (\mathsf{end} \lor \bigcirc \neg\neg a) \land \neg\mathsf{end}$. Then TAUT gives $\vdash \bullet a \land \neg\mathsf{end} \to \bigcirc \neg\neg a \land \neg\mathsf{end}$. Then, two applications of COMMNEGNEXT give $\vdash \bullet a \land \neg\mathsf{end} \to \neg\neg \bigcirc a \land \neg\mathsf{end}$. Then NEXTDISTRLEM follows by double-negation elimination from TAUT. ✓.

Now, consider each direction separately

($\to$) TAUT gives $\vdash (\bigcirc a \to \bigcirc b) \to \mathsf{end} \lor (\neg \bigcirc a \lor \bigcirc b) \land \neg\mathsf{end}$. We can distribute, using TAUT, and get $\vdash (\bigcirc a \to \bigcirc b) \to \mathsf{end} \lor (\neg \bigcirc a \land \neg\mathsf{end}) \lor (\neg\neg \bigcirc b \land \neg\mathsf{end}$. Finally, we can apply COMMNEGNEXT to both applicable disjunctive clauses, and get $\vdash (\bigcirc a \to \bigcirc b) \to \mathsf{end} \lor$

$(\bigcirc\neg a \wedge \neg\mathsf{end}) \vee (\bullet\, b \wedge \neg\mathsf{end})$. Then we can factor the $(\neg\mathsf{end})$ out, and desugar the disjunction to implication, giving $\vdash (\bigcirc\, a \to \bigcirc\, b) \to \mathsf{end} \vee ((\bullet\, a \to \bullet\, b) \wedge \neg\mathsf{end})$. Then, using WkNextDistr, we get $\vdash (\bigcirc\, a \to \bigcirc\, b) \to \mathsf{end} \vee (\bullet(a \to b) \wedge \neg\mathsf{end})$. Then, nextdistrlem applies to give $\vdash \bigcirc\, a \to \bigcirc\, b \to \mathsf{end} \vee (\bigcirc(a \to b) \wedge \neg\mathsf{end})$. The result follows by Taut.

($\leftarrow$) Let $\phi = \bigcirc(a \to b) \vee \mathsf{end}$. We know by Taut that $\vdash \phi \to \bigcirc(a \to b) \wedge \mathsf{end} \vee \bigcirc(a \to b) \wedge \neg\mathsf{end} \vee \mathsf{end}$. We can simplify this formula using Taut and EndNextContra, and see that $\vdash \phi \to \bigcirc(a \to b) \wedge \neg\mathsf{end} \vee \mathsf{end}$. Then we can apply nextdistrlem to get $\vdash \phi \to \neg\mathsf{end} \wedge \bullet(a \to b) \vee \mathsf{end}$. Then WkNextDistr gives $\vdash \phi \to \neg\mathsf{end} \wedge \bullet\, a \to \bullet\, b \vee \mathsf{end}$. Now, we use Taut to get $\vdash \phi \to (\neg\bullet\, a \wedge \neg\mathsf{end} \vee \bullet\, b \wedge \neg\mathsf{end} \vee \mathsf{end}$. Now we use CommNegNext 3 times to get $\vdash \phi \to (\bigcirc\, a \to \bigcirc\, b) \wedge \neg\mathsf{end} \vee \mathsf{end}$. And now, we can use EndNextContra to get $\vdash \phi \to (\bigcirc\, a \to \bigcirc\, b) \wedge \neg\mathsf{end} \vee \mathsf{end} \wedge (\neg\bigcirc\, a)$ and finally, Taut gives $\vdash \phi \to (\bigcirc\, a \to \bigcirc\, b) \wedge \neg\mathsf{end} \vee \mathsf{end} \wedge (\bigcirc\, a \to \bigcirc\, b)$. Then Taut gives $\vdash \bigcirc(a \to b) \vee \mathsf{end} \to \bigcirc\, a \to \bigcirc\, b$.

$\triangle$

Lemma A.25. *The following rule can be derived from the rules in Definition 3.19*

$$\frac{\vdash a \to b}{\vdash \bigcirc\, a \to \bigcirc\, b} \qquad \text{(NextMonotone)}$$

Proof. We know $\vdash a \to b$, we want to show $\vdash \bigcirc\, a \to \bigcirc\, b$. WkNextStep gives us $\vdash \bullet(a \to b)$, to which we add a Taut to get $\vdash \bullet(a \to b) \wedge \mathsf{end} \vee \bullet(a \to b) \wedge \neg\mathsf{end}$. We can apply EndNextContra and Taut to get

$$\vdash (\neg\bigcirc\, a \vee \bigcirc\, b) \wedge \bullet(a \to b) \wedge \neg\mathsf{end}$$

Then CommNegNext turns $\bullet(a \to b) \wedge \neg\mathsf{end}$ into $\bigcirc(a \to b) \wedge \neg\mathsf{end}$. Then NextDistr gives $\vdash (\neg\bigcirc\, a \vee \bigcirc\, b) \wedge (\bigcirc\, a \to \bigcirc\, b)$, so we conclude $\vdash \bigcirc\, a \to \bigcirc\, b$ by Taut.

$\triangle$

Lemma A.26. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \bullet\, a \leftrightarrow (\bigcirc\, a \vee \mathsf{end}) \qquad \text{(NextWkNext)}$$

Proof. We know from TAUT, that $\vdash \bullet a \leftrightarrow \bullet a \wedge (\mathsf{end} \vee \neg \mathsf{end})$. Then by TAUT, COMMNEGNEXT, and TAUT again, we get $\vdash \bullet a \leftrightarrow (\bullet a \wedge \mathsf{end}) \vee \bullet \neg a$. Then we see that $\vdash \bullet a \wedge \mathsf{end} \vee \bigcirc \neg a \rightarrow \mathsf{end} \vee \bigcirc \neg a$ by TAUT, and the converse by ENDNEXTCONTRA. Then, by TAUT, conclude that $\vdash \bullet a \leftrightarrow (\bigcirc a \vee \mathsf{end})$.

△

Lemma A.27. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \bullet(a \wedge b) \leftrightarrow \bullet a \wedge \bullet b \qquad \textit{(}\text{WKNEXTANDDISTR}\textit{)}$$

Proof.

$\vdash \bullet(a \rightarrow b)$	
$\leftrightarrow \bullet(\neg(a \rightarrow \neg b))$	def.
$\leftrightarrow \bullet(\neg(a \rightarrow \neg b)) \wedge \mathsf{end} \vee \neg \bigcirc(a \rightarrow \neg b) \wedge \neg \mathsf{end}$	TAUT
$\leftrightarrow \mathsf{end} \vee \neg \bigcirc(a \rightarrow \neg b) \wedge \neg \mathsf{end}$	ENDNEXTCONTRA,TAUT
$\leftrightarrow \mathsf{end} \vee \neg \bullet(a \rightarrow \neg b) \wedge \neg \mathsf{end}$	NEXTWKNEXT
$\leftrightarrow \mathsf{end} \vee \neg(\bullet a \rightarrow \bullet \neg b) \wedge \neg \mathsf{end}$	WKNEXTDISTR
$\leftrightarrow \mathsf{end} \vee \neg(\bullet a \rightarrow \neg \bullet b) \wedge \neg \mathsf{end}$	COMMNEGNEXT
$\leftrightarrow \mathsf{end} \vee (\bullet a \wedge \bullet b) \wedge \neg \mathsf{end}$	TAUT
$\leftrightarrow \bullet a \wedge \bullet b \wedge \mathsf{end} \vee \bullet a \wedge \bullet b \wedge \neg \mathsf{end}$	ENDNEXTCONTRA,TAUT
$\leftrightarrow \bullet a \wedge \bullet b$	TAUT

So conclude $\vdash \bullet(a \wedge b) \leftrightarrow (\bullet a \wedge \bullet b)$

△

Lemma A.28. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \Box a \leftrightarrow a \wedge \bullet \Box a \qquad \textit{(}\text{ALWUNROLL}\textit{)}$$

Proof. Desugar the proposition to $\vdash (a \mathcal{W} \bot) \leftrightarrow a \wedge \bullet(a \mathcal{W} \bot)$. Then WKUNTILUNROLL gives $\vdash a \mathcal{W} \bot \leftrightarrow \bot \vee (a \wedge \bullet(a \mathcal{W} \bot))$. Then TAUT lets you cancel the disjunctive $\bot$, so conclude $\vdash a \mathcal{W} \bot \leftrightarrow a \wedge \bullet(a \mathcal{W} \bot)$, which resugars to $\vdash \Box a \leftrightarrow a \wedge \bullet \Box a$. △

Lemma A.29. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash (\square\, a \wedge \neg \mathsf{end}) \leftrightarrow a \wedge \bigcirc \square\, a \qquad (\textsc{AlwUnrollStep})$$

Proof. We know $\vdash (\square\, a \wedge \neg \mathsf{end}) \leftrightarrow (a \wedge \bullet \square\, a) \wedge \neg \mathsf{end}$ by AlwUnroll and Taut. Then NextWkNext gives $\vdash (a \wedge \bullet \square\, a \wedge \neg \mathsf{end}) \rightarrow a \wedge \bigcirc \square\, a$. Similarly, $\vdash a \wedge \bigcirc \square\, a \rightarrow a \wedge \bullet \square\, a \wedge \neg \mathsf{end}$ follows from NextNotEnd and NextWkNext. This gives $\vdash (\square\, a \wedge \neg \mathsf{end}) \leftrightarrow a \wedge \bigcirc \square\, a$.

△

Lemma A.30. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash (\square\, a \wedge \mathsf{end}) \rightarrow a \qquad (\textsc{AlwEnd})$$

Proof. AlwUnroll gives $\vdash (\square\, a \wedge \mathsf{end}) \leftrightarrow a \wedge \bullet \square\, a$. Then Taut gives $\vdash a \wedge \bullet \square\, a \rightarrow a$. Compose these to get $\vdash (\square\, a \wedge \mathsf{end}) \rightarrow a$.

△

Lemma A.31. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \lozenge\, a \leftrightarrow a \vee \bigcirc \lozenge\, a \qquad (\textsc{EverUnroll})$$

Proof. By definition $\vdash \lozenge\, a \leftrightarrow \neg \square \neg a$. Then we can apply AlwUnroll which gives $\vdash \lozenge\, a \leftrightarrow \neg(\neg a \wedge \bullet \square \neg a)$. Then Taut shows us that $\vdash \lozenge\, a \leftrightarrow a \vee \bigcirc \neg \square \neg a$. Then again by definition, $\vdash \lozenge\, a \leftrightarrow a \vee \bigcirc \lozenge\, a$. △

Lemma A.32. *The following rule can be derived from the rules in Definition 3.19*

$$\frac{\vdash a \rightarrow b}{\vdash \square\, a \rightarrow \square\, b} \qquad (\textsc{AlwMonotone})$$

Proof. Assume $\vdash a \rightarrow b$, to show $\vdash \square\, a \rightarrow \square\, b$. Apply AlwUnroll one step to get $\vdash \square\, a \leftrightarrow a \wedge \bullet \square\, a$. Then applying the assumption and Taut gives $\vdash \square\, a \rightarrow b \wedge \bullet \square\, a$. So we can conclude by Taut that $\vdash \square\, a \rightarrow \bullet \square\, a$ and $\vdash \square\, a \rightarrow b$, which allows us to derive $\vdash \square\, a \rightarrow \square\, b$ via Induction.

△

Lemma A.33. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \neg(\neg \bigcirc \top \wedge \bigcirc a) \qquad (\textsc{EndNextContra'})$$

Proof. NextNotEnd says $\vdash \mathsf{end} \to \neg \bigcirc a$. By DeMorgan's laws in Taut, conclude $\vdash \neg(\mathsf{end} \wedge \bigcirc a)$.

△

Lemma A.34. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash (\mathsf{end} \wedge \Diamond a) \to a \qquad (\textsc{EverEnd})$$

Proof. We know $\vdash (\mathsf{end} \wedge \Diamond a) \leftrightarrow ((\mathsf{end} \wedge a) \vee (\mathsf{end} \wedge \bigcirc \Diamond a))$ from EverUnroll. Then since $\vdash \neg(\mathsf{end} \wedge \bigcirc \Diamond a)$ by EndNextContra', conclude that $\vdash (\mathsf{end} \wedge \Diamond a) \leftrightarrow (\mathsf{end} \wedge a)$. And finally $\vdash (\mathsf{end} \wedge \Diamond a) \to a$ by Taut.

△

Lemma A.35. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \Box a \wedge \Box b \to \Box(a \wedge b) \qquad (\textsc{AlwaysAndDistr})$$

Proof. The AlwUnroll and Taut rules give $\vdash \Box a \wedge \Box b \to a \wedge b \wedge \bullet \Box a \wedge \bullet \Box b$. Then apply WkNextAndDistr and Taut to get $\vdash \Box a \to (a \wedge b)$ and $\vdash \Box a \to \bullet(\Box a \wedge \Box b)$. Then an application of Induction proves our goal, that $\vdash \Box a \to \Box(a \wedge b)$.

△

Lemma A.36. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \Box a \wedge \Diamond b \to \Diamond(b \wedge a) \qquad (\textsc{AlwaysEver})$$

Proof. Taut gives us that $\vdash \Box a \wedge \Diamond b \leftrightarrow (\Box a \wedge \Diamond b) \wedge (\neg \Diamond(b \wedge a) \vee \Diamond(b \wedge a))$. Notice that $\vdash \Box a \wedge \neg \Diamond(b \wedge a) \leftrightarrow \Box((\neg b \vee \neg a) \wedge a)$ by AlwaysAndDistr, then Taut and AlwaysAndDistr lets us conclude that $\vdash (\Box a \wedge \neg \Diamond(b \wedge a)) \leftrightarrow \Box(\neg b) \wedge \Box a$. So, now we can say that $\vdash \Box a \wedge \Diamond b \wedge \neg \Diamond(b \wedge a) \leftrightarrow \Box a \wedge \Box(\neg b) \wedge \Diamond b$, which is contradictory by Taut.

Conclude that $\vdash \Box a \wedge \Diamond b \to \Diamond(b \wedge a))$, by Taut.

△

Lemma A.37. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \Box\, a \to \Diamond(\mathsf{end} \wedge a) \qquad (\textsc{AlwaysFinite})$$

Proof. We know $\vdash \Box\, a \leftrightarrow (\Box\, a \wedge \Diamond\, \mathsf{end})$ by FINITE and TAUT. Then ALWAYSEVER and TAUT give $\vdash \Box\, a \to \Diamond(\mathsf{end} \wedge a)$. △

Lemma A.38. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \bigcirc(a \vee b) \leftrightarrow \bigcirc\, a \vee \bigcirc\, b \qquad (\textsc{NextOrDistr})$$

Proof. We want to show $\vdash \bigcirc(a \vee b) \leftrightarrow \bigcirc\, a \vee \bigcirc\, b$. We can desugar this to $\vdash \bigcirc((\neg a) \to b) \leftrightarrow \neg \bigcirc\, a \to \bigcirc\, b$.

($\to$) The rules NEXTDISTR and NEXTNOTEND give $\vdash \bigcirc((\neg a) \to b) \to \neg\mathsf{end} \wedge (\bigcirc(\neg a) \to \bigcirc\, b))$. Then by TAUT and COMMNEGNEXT, we get $\vdash \bigcirc((\neg a) \to b) \to (\bigcirc \neg a) \to \bigcirc\, b$

($\leftarrow$) We know that $\vdash \neg \bigcirc\, a \to \bigcirc\, b \leftrightarrow (\neg \bigcirc\, a \to \bigcirc\, b) \wedge (\neg \bigcirc\, a \vee \bigcirc\, a)$. Then we also see, by TAUT, that $\vdash \neg \bigcirc\, a \to \bigcirc\, b \to (\neg \bigcirc\, a \to \bigcirc\, b) \wedge (\bigcirc\, b \vee \bigcirc\, a)$ which lets us conclude by NEXTNOTEND and TAUT $\vdash (\neg \bigcirc\, a \to \bigcirc\, b) \to (\neg \bigcirc\, a \to \bigcirc\, b) \wedge \neg\mathsf{end}$. Then COMMNEGNEXT and TAUT give $\vdash (\neg\mathsf{end} \wedge (\neg \bigcirc\, a \to \bigcirc\, b)) \to (\bigcirc \neg a \to \bigcirc\, b)$. Then by NEXTDISTR we can see $\vdash (\neg\mathsf{end} \wedge (\neg \bigcirc\, a \to \bigcirc\, b)) \to \bigcirc(\neg a \to b)$ which ultimately lets us conclude that $\vdash (\bigcirc\, a \vee \bigcirc\, b) \to \bigcirc(a \vee b)$.

△

Lemma A.39. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \bigcirc(a \wedge b) \leftrightarrow \bigcirc\, a \wedge \bigcirc\, b \qquad (\textsc{NextAndDistr})$$

Proof. We will phrase this in terms of distributivity over $\vee$. So we can rephrase the goal as $\vdash \bigcirc(\neg(\neg a \vee \neg b)) \leftrightarrow \neg(\neg \bigcirc\, a \vee \neg \bigcirc\, b)$. We will prove each case separately.

By NEXTNOTEND and TAUT, we get $\vdash \bigcirc(\neg(\neg a \vee \neg b)) \leftrightarrow \bigcirc(\neg(\neg a \vee \neg b)) \wedge \neg\mathsf{end}$. Then from COMMNEGNEXT and TAUT, we see $\vdash \bigcirc(\neg(\neg a \vee$

$\neg b)) \leftrightarrow \neg(\bigcirc(\neg a \vee \neg b)) \wedge \neg\mathsf{end}$ Then by NegNextDistr, we get $\vdash \bigcirc(\neg(\neg a \vee \neg b)) \leftrightarrow \neg(\bigcirc \neg a \vee \bigcirc \neg b) \wedge \neg\mathsf{end}$ And finally by CommNegNext, and Taut we conclude that $\vdash \bigcirc(\neg(\neg a \vee \neg b)) \leftrightarrow \neg(\neg \bigcirc a \vee \neg \bigcirc b) \wedge \neg\mathsf{end}$. Then finally by NextNotEnd and Taut, we conclude $\vdash \bigcirc(a \wedge b) \leftrightarrow \bigcirc a \wedge \bigcirc b$.

△

Lemma A.40. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \bullet \top \qquad \text{(WkNextTop)}$$

Proof. Taut gives $\vdash \top$. Then from WkNextStep, we get $\vdash \bullet \top$. △

Lemma A.41. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash (a\,\mathcal{U}\,b) \leftrightarrow b \vee (a \wedge \bigcirc(a\,\mathcal{U}\,b)) \qquad \text{(UntilUnroll)}$$

Proof. We prove both directions simultaneously. We show $(a\,\mathcal{U}\,b) \rightarrow b \vee a \wedge \bigcirc(a\,\mathcal{U}\,b)$ using only "if and only if" implications.

Assume $a\,\mathcal{U}\,b$, to show $b \vee a \wedge \bigcirc(a\,\mathcal{U}\,b)$. Desugar both the premise to get $a\,\mathcal{W}\,b \wedge \Diamond b$. Apply WkUntilUnroll and EverUnroll to get the statement $(b \vee a \wedge \bullet(a\,\mathcal{W}\,b)) \wedge (b \vee \bigcirc \Diamond b)$. We can simplify this using Taut, and we see that $b \vee a \wedge \bullet(a\,\mathcal{W}\,b) \wedge \bigcirc \Diamond b$.

We can introduce the tautology $\neg\mathsf{end} \vee \mathsf{end}$ to the right branch of the $\vee$, giving $(b \vee a \wedge \bullet(a\,\mathcal{W}\,b) \wedge \bigcirc \Diamond b) \wedge (\mathsf{end} \vee \neg\mathsf{end})$. Expanding this, we can see that end and $\bigcirc \Diamond b$ result in a contradiction via NextNotEnd. So we simply have $b \vee a \wedge \bullet\, a\,\mathcal{W}\,b \wedge \bigcirc \Diamond b \wedge \neg\mathsf{end}$. Since $\neg\mathsf{end}$, we note that $\bullet$ can be replaced with $\bigcirc$ (via CommNegNext). So we get $b \vee a \wedge \bigcirc(a\,\mathcal{W}\,b) \wedge \bigcirc \Diamond b$. Then using NextAndDistr, we get $b \vee (a \wedge \bigcirc(a\,\mathcal{W}\,b \wedge \Diamond b))$.

△

Lemma A.42. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \neg(a\,\mathcal{U}\,b) \leftrightarrow (\neg b)\,\mathcal{W}\,(\neg a \wedge \neg b) \qquad \text{(NotUntil)}$$

Proof. We will prove each direction separately.

($\rightarrow$) The UntilUnroll rule gives $\vdash \neg(a\,\mathcal{U}\,b) \leftrightarrow \neg(b \vee a \wedge \bigcirc(a\,\mathcal{U}\,b)$. Then Taut gives us $\vdash \neg(a\,\mathcal{U}\,b) \leftrightarrow (\neg b \wedge \neg a) \vee (\neg b \wedge \neg \bigcirc(a\,\mathcal{U}\,b))$. Note that Taut gives $\vdash \neg(a\,\mathcal{U}\,b) \rightarrow \neg b$. This will be useful later. Consider each case separately:

Notice that we can immediately turn the $(\neg b \wedge \neg a)$ into $(\neg b)\;\mathcal{W}\;(\neg a \wedge \neg b)$ from WkUntilUnroll and Taut. So, we now have $\vdash \neg(a\;\mathcal{U}\;b) \leftrightarrow ((\neg b)\;\mathcal{W}\;(\neg a \wedge \neg b)) \vee (\neg b \wedge \neg \bigcirc(a\,\mathcal{U}\,b))$.

We can apply CommNegNext to conclude that

$$\vdash \neg(a\,\mathcal{U}\,b) \leftrightarrow ((\neg b)\;\mathcal{W}\;(\neg a \wedge \neg b)) \vee (\neg b \wedge \bigcirc\neg(a\,\mathcal{U}\,b)) \vee (\neg b \wedge \mathsf{end}).$$

Then EndNextContra gives $\vdash \neg b \wedge \mathsf{end} \rightarrow \neg b \wedge \bullet \Box \neg b$, which allows us to conclude the following from AlwaysWkUntil and Taut:

$$\vdash \neg(a\,\mathcal{U}\,b) \rightarrow ((\neg b)\;\mathcal{W}\;(\neg a \wedge \neg b)) \vee (\neg b \wedge \mathsf{end}).$$

Further, we can show, by NextWkNext that $\vdash \neg(a\,\mathcal{U}\,b) \wedge (a \vee b) \rightarrow \neg \wedge \bullet(\neg(a\;\mathcal{W}\;b))$, which by induction gives $\Box \neg b$, which in turn by AlwaysWkUntil and Taut derives $\vdash \neg(a\,\mathcal{U}\,b) \rightarrow (\neg b)\;\mathcal{W}\;(\neg a \wedge \neg b)$.

($\leftarrow$) WkUntilUnroll gives $\vdash (\neg b)\;\mathcal{W}\;(\neg b \wedge \neg b) \leftrightarrow (\neg a \wedge \neg b) \vee (\neg b \wedge \bullet((\neg b)\;\mathcal{W}\;(\neg a \wedge \neg b)))$.

We can introduce $\neg(a\;\mathcal{U}\;b) \vee (a\;\mathcal{U}\;b)$ to the first disjunct. However $\vdash a\;\mathcal{U}\;b \rightarrow a \vee b$, by WkUntilUnroll which is contradictory, so conclude

$$\vdash (\neg b)\;\mathcal{W}\;(\neg b \wedge \neg b) \rightarrow a\;\mathcal{W}\;b \vee (\neg b \wedge \bullet((\neg b)\;\mathcal{W}\;(\neg a \wedge \neg b)))$$

Then Taut gives

$\vdash ((\neg b)\;\mathcal{W}\;(\neg b \wedge \neg b) \wedge (a \vee b) \rightarrow (\neg b \wedge \bullet((\neg b)\;\mathcal{W}\;(\neg a \wedge \neg b) \wedge (a \vee b)))$,

to which we can apply Induction and conclude

$$\vdash (\neg b)\;\mathcal{W}\;(\neg b \wedge \neg b) \wedge (a \vee b) \rightarrow \Box \neg b$$

Then, finally, we have

$$\vdash (\neg b)\;\mathcal{W}\;(\neg a \wedge \neg b) \rightarrow \neg(a\;\mathcal{W}\;b) \vee \Box \neg b$$

This is equivalent to the result by Taut, because $\vdash \neg(a\;\mathcal{W}\;b) \vee \Box \neg b \leftrightarrow \neg(a\,\mathcal{U}\,b)$ is a result of syntactic sugarings and Taut.

$\triangle$

Lemma A.43 (Duality of $\Diamond$ and $\Box$)**.** *The following rules can be derived from the rules in Definition 3.19*

$$\vdash \Box\, a \leftrightarrow \neg \Diamond \neg a \qquad (\textsc{AlwaysEverDual})$$

$$\vdash \Diamond\, a \leftrightarrow \neg \Box \neg a \qquad (\textsc{EverAlwaysDual})$$

Proof.

1. Desugar $\Diamond\, a$ to $\neg \Box \neg a$. Then the goal is $\Box\, a \leftrightarrow \neg\neg \Box \neg\neg a$. The negations cancel by Taut, and the result follows.

2. By definition.

$\triangle$

Lemma A.44. *The following rule can be derived from the rules in Definition 3.19*

$$\frac{\vdash a \to \bullet\, a \wedge b}{\vdash a \to \Box\, b} \qquad (\textsc{Induction'})$$

Proof. Assume $\vdash a \to \bullet\, a \wedge b$. Taut breaks this down to $\vdash a \to \bullet\, a$ and $\vdash a \to b$. Then we apply Induction and conclude $\vdash a \to \Box\, b$. $\triangle$

Lemma A.45. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash \Box\, a \to a \,\mathcal{W}\, b \qquad (\textsc{AlwaysWkUntil})$$

Proof. We know from Taut that $\vdash \Box\, a \to ((a \,\mathcal{W}\, b) \vee \neg(a \,\mathcal{W}\, b))$, so if we show that $\vdash \neg(\Box\, a \wedge \neg(a \,\mathcal{W}\, b))$, we can conclude that $\Box\, a \to (a \,\mathcal{W}\, b)$.

We know by AlwUnroll and WkUntilUnroll that

$$\vdash \Box\, a \to a \wedge \bullet \Box\, a, \text{ and}$$

$$\vdash (\neg a \,\mathcal{W}\, b) \to (\neg b \wedge \neg a \vee \neg b \wedge \circ \neg(a \,\mathcal{W}\, b))$$

Then, we can combine these two with Taut and NextWkNext to see that

$$\vdash (\Box\, a \wedge \neg(a \,\mathcal{W}\, b)) \to \bullet \Box\, a \wedge \bullet \neg(a \,\mathcal{W}\, b)$$

Then WKNEXTANDDISTR gives $\vdash (\Box\, a \wedge \neg(a\, \mathcal{W}\, b)) \to \bullet\, \Box\, a \wedge \bullet\, \neg(a\, \mathcal{W}\, b)$ and TAUT admits $\vdash (\Box\, a \wedge \neg(a\, \mathcal{W}\, b)) \to (\Box\, a \wedge \neg(a\, \mathcal{W}\, b))$. Then INDUCTION lets us conclude that

$$\vdash (\Box\, a \wedge \neg(a\, \mathcal{W}\, b)) \to \Box(\Box\, a \wedge \neg(a\, \mathcal{W}\, b))$$

Then by ALWUNROLL and TAUT we have

$$\vdash (\Box\, a \wedge \neg(a\, \mathcal{W}\, b)) \leftrightarrow \Box(\Box\, a \wedge \neg(a\, \mathcal{W}\, b)).$$

Now, we from above that $\vdash \neg(a\, \mathcal{W}\, b) \to \bigcirc\, \neg(a\, \mathcal{W}\, b)$, which by NEXTNOTEND gives $\vdash \neg(a\, \mathcal{W}\, b) \to \not\,\mathsf{end}$. Then by ALWMONOTONE and TAUT, conclude that

$$\vdash \Box(\Box\, a \wedge \neg(a\, \mathcal{W}\, b)) \to \Box\, \neg\mathsf{end}$$

which, by TAUT shows $\vdash \neg(\Box\, a \wedge \neg(a\, \mathcal{W}\, b))$, so conclude that $\vdash \Box\, a \to a\, \mathcal{W}\, b$ △

Lemma A.46. *The following rule can be derived from the rules in Definition 3.19*

$$\frac{\vdash c \to b \vee (a \wedge \bullet\, c)}{\vdash c \to a\, \mathcal{W}\, b} \qquad \text{(WKUNTILINDUCTION)}$$

Proof. Let $d \triangleq c \wedge \neg a\, \mathcal{W}\, b$. Recall the WKUNTILUNROLL, rule, which says that $\vdash a\, \mathcal{W}\, b \to b \wedge [a \wedge \bullet(a\, \mathcal{W}\, b)]$. Its negation is $\vdash \neg(a\, \mathcal{W}\, b) \to \neg b \wedge [\neg a \vee \bigcirc\, \neg(a\, \mathcal{W}\, b)]$. Then we can say $\vdash d \to \neg b \wedge [\neg a \vee \bullet\, \neg(a\, \mathcal{W}\, b)]$, by NEXTWKNEXT. Then because $\vdash d \to c$, TAUT gives that $\vdash d \to a \wedge \bullet\, c \wedge \neg b \wedge \bigcirc\, \neg(a\, \mathcal{W}\, b)$. Note especially that $\vdash d \to a$, and $\vdash d \to \bullet\, c$. Hence, also, $\vdash d \to \bullet\, c \wedge \bullet\, \neg(a\, \mathcal{W}\, b)$. Then using WKNEXTANDDISTR, we see that $\vdash d \to \bullet[c \wedge \neg(a\, \mathcal{W}\, b)]$, which is, by definition, $\vdash d \to \bullet\, d$. Since we have $\vdash d \to a$ and $\vdash d \to \bullet\, d$, conclude $\vdash d \to \Box\, a$ by INDUCTION.

We know $\vdash d \to \Box\, a$, so by ALWAYSWKUNTIL, we get $\vdash d \to a\, \mathcal{W}\, b$. This is equivalent to $\vdash [c \wedge \neg(a\, \mathcal{W}\, b)] \to a\, \mathcal{W}\, b$, which entails $\vdash \neg(c \wedge \neg(a\, \mathcal{W}\, b))$. Then conclude $\vdash c \to a\, \mathcal{W}\, b$ via TAUT. △

Lemma A.47. *The following rule can be derived from the rules in Definition 3.19*

$$\vdash (a\, \mathcal{W}\, b \wedge \Box\, \neg b) \to \Box\, a \qquad \text{(WKUNTILALWAYS)}$$

Proof. WkUntilUnroll and AlwUnroll give $\vdash\ (a\ \mathcal{W}\ b) \wedge \Box\neg b) \to (b \vee a \wedge \bullet(a\ \mathcal{W}\ b)) \wedge \neg b \wedge \bullet\Box\neg b$. Then Taut and WkNextAndDistr gives $\vdash (a\ \mathcal{W}\ b) \wedge \Box\neg b) \to b \vee (\neg b \wedge a \wedge \bullet(a\ \mathcal{W}\ b \wedge \neg b)$, and then Taut again gives $\vdash (a\ \mathcal{W}\ b \vee \wedge\neg b) \to \bot \vee a \wedge \bullet((a\ \mathcal{W}\ b) \wedge \Box\neg b)$. Then applying Taut and WkUntilInduction gives $\vdash (a\ \mathcal{W}\ b \wedge \Box\neg b) \to a\ \mathcal{W}\ \bot$, which is, by definition, exactly what we wanted to prove. △