Technical Report: Decidable Fragments of Matching Logic

Nishant Rodrigues, Xiaohong Chen and Grigore Roşu
University of Illinois at Urbana-Champaign
{nishant2,xc3,grosu}@illinois.edu

February 2, 2021

Abstract

Matching logic is a unifying logic aimed at defining programming language semantics, and reasoning about various program and language properties. It is a general logic designed with minimalism in mind. With only eight syntactic constructs, matching logic can define many important logical frameworks and languages as its theories. Yet, to our knowledge, no research has been conducted into the decidabiltiy of matching logic. In this paper, we begin such an initiative with respect to decidable fragments of matching logic and identify the first non-trivial decidable fragment for the empty theory. Our decision procedure extends a tableau system for modal μ-calculus. We also give an implementation of the proposed decision procedure and show that with modifications, it can be extended to support theories with certain axioms.

1 Introduction

Matching logic [1] was developed as the mathematical foundation [2] of the $\mathbb{K}$ framework [3] — a rewriting-based framework for defining and reasoning about the formal semantics of programming languages. It is therefore intended to be expressive enough to encompass both abstractions for defining programing language semantics, as well as tools for model checking and verification.

Matching logic adopts a minimal design [4]. Its formulae, called *patterns*, are built from constants, applications, logical operators, quantification, and fixedpoints. Unlike FOL, matching logic makes no distinction between terms and formulae giving matching logic the flexibility to subsume various syntaxes, including predicates, assertions, expressions, etc. of FOL [1], modal logics (computation tree logic (CTL) [5], linear temporal logic (LTL) [5], propositional dynamic logic (PDL) [5], modal μ-calculus [5], . . .), separation logic [6], reachability logic [5], algebraic structures modulo axioms [7], and type systems [8]. This syntactic generality allows for uniformly specifying and reasoning about properties typically expressed in disparate logical systems in their original notation, within single logical framework yet without awkward encodings.

We consider these "awkwardness-free" encodings a big selling point for matching logic. Indeed, a logical framework may work in principle, albeit with a large gap between the target logic and its encoding in the framework. Yet, despite its fairly minimal design, matching logic is able to capture each of these logics with little or no representational distance.

Matching logic becomes a *lingua franca* within which we may talk about properties of the mentioned logics, or develop procedures not limited to specific ones. The need for such generality is clearly justified by the similarities and redundancies between, for example, the tableau-based decision procedures for propositional logic [9], modal μ-calculus [10], for LTL [11], FOL with equality [12] to name a few. Indeed, there already exist initiatives for such logic-generic procedures using matching logic as their basis. In [13], the authors develop a prototype proof framework for reasoning about fixedpoints. A small set of proof strategies are derived that are sufficient to prove separation logic, LTL, and reachability formulae. However, no thought is given to decidability.

In this paper, we study the decidability of matching logic. By decidability, we refer to the decidability of the following *satisfiability* problem, stated informally:

> Given an axiom set Γ and a pattern φ, does there exist a model M and an assignment of variables ρ, such that M validates Γ and there exists an element $e \in |\varphi|_\rho$, meaning that e is *matched* by φ in M.

This closesly parallels the satisfiability modulo theories problem of FOL:

> Given an axiom set Γ and a formula φ, does there exist a model M and an assignment of variables ρ, such that M validates Γ and φ.

Posed by Hilbert and Ackermann as a fundamental challenge of mathematics (the Entscheidungsproblem) in 1928 [14], finding an algorithm to solve this problem was proved impossible for arbitrary mathematical statements and theories independently by Church [15] and Turing [16] in 1936. However, this has not caused the problem to disappear. Instead it has become one of classification — for which classes of mathematical statements can we find such an algorithm?

Much progress has been made since then, and continues to be made, in this active area of research. Various theories and syntactic fragments (e.g. quantifier prefixes) of FOL have decision procedures [17]. Several other logics, such as propositional modal logic [18], modal μ-calculus [10], linear temporal logic (LTL) [11], CTL* [19], guarded fixedpoint logic [20], and more have been proved decidable. In addition, several other mathematical abstractions not typically considered logics, such as unification modulo axioms [21], have also been shown decidable.

Many of these decision procedures and proofs exist in their own private universes with varied syntaxes, semantics, models and proof rules. Yet, most of these logics and abstractions may be defined naturally as theories within matching logic. This leads us to wonder, is matching logic a tool we may leverage to find commonalities between these assorted decision procedures? Can it help up expand these fragments? Does it give us insight into what properties make these fragments decidable, and what makes them undecidable?

While important and interesting questions in their own right, these questions are not of purely theoretic motivation. Let us now return to matching logic's origin, as the logical foundations of the $\mathbb{K}$ framework. This framework subscribes a "sematics-first" philosophy, shown in Figure 1. Under this philosophy, programming language development should start with a formal mathematical semantics. From this formal semantics, we may then derive the various tools for programming languages, such as compilers, interpreters, model checkers, verification tools, etc. This approach has proven effective and practical, even for complex languages in varied paradigms, such as C [22], Haskell [23], Etherium Virtual Machine [24], to name a few. Language semantics and specifications written in $\mathbb{K}$ are translated to matching logic theories. $\mathbb{K}$'s various tooling are considered best-effort implementations to prove the validity of these specifications. Knowing the limits of decidability in matching logic will inform us about how we may use these tools more effectively, and where we need to ask the user for additional lemmas, assumptions and annotations.

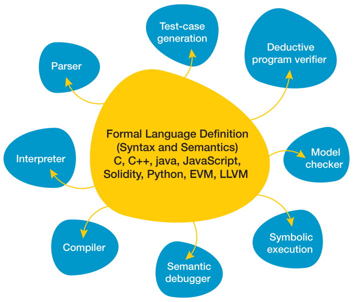


Figure 1: The semantics-first approach taken by $\mathbb{K}$.

In this work, we make a first foray into such an effort by generalizing a decision procedure for modal μ-calculus to a fragment of matching logic. In Section 2, we discuss related research. In Section 3, we present a formal introduction to matching logic. In Section 4, we look at the existing results about decidability in matching logic. The next three sections present our main results:

- In Section 5, we introduce a decision procedure for a fragment of matching logic.
- In Section 6, we show how we can take theories into account by generalizing the previous decision procedure to LTL.
- Finally, in Section 7 we briefly review out implementation of the decision procedure.

2 Related Work

In this section, we briefly introduce works we generalize, or with similar goals.

2.1 Tableau Systems for Modal Logics

Various modal logics have such as LTL, CTL, modal logic, modal μ-calculus have tableau systems for the satisfiability of their formulae. The tableau system presented in this paper in fact generalizes that of modal μ-calculus, and with some modifications, subsumes that of LTL.

2.2 Monadic Second Order on Graphs with Bounded Treewidth

Monadic second order logic is a logic that allows quantification over both sets and elements. It has been shown that when restricted to models that are graphs with bounded tree width, formulae in this logic may be decided in linear time [25].

2.3 Guarded Logics

Many of the above modal logics have been shown to have translations into guarded logics, or guarded fixedpoint logics. Guarded logics restrict quantification so that they can only reference elements that are closely related. This is done syntactically, by restricting quantification to formulae of the form.

$$\exists \bar{y}.(\alpha(\bar{x}, \bar{y}) \wedge \psi(\bar{x}, \bar{y}) \text{ or } \forall \bar{y}.(\alpha(\bar{x}, \bar{y}) \to \psi(\bar{x}, \bar{y})$$

where α is a “guard” that must mention all free variables in ψ. Depending on which guarded logic is considered, α may have different restrictions placed on it. As originally proposed in [20], α was restricted to atomic propositions. Whereas, loosely guarded logics [26] generalize this to conjunctions of atoms, such that for each pair x, x' of variables one in $\bar{x} \cup \bar{y}$, the other quantified in $\bar{y}$ an atomic proposition in the guard mentions both x and x'.

While the fragment of matching logic we build the decision procedure for may likely be translated to loosely guarded fixedpoint logic, matching logic offers other selling points such as the low representational distance and the possibility of restricting models using theories.

2.4 A Unified Proof Framework

In [13], the authors develop a prototype proof framework for matching logic. It deals with the validity of matching logic patterns, a property closely related to satisfiability. The prototype is aimed at handling fixedpoints generically across a set of matching logic theories, specifically separation logic, LTL and reachability. A small set of strategies derived from the matching logic proof system, that are sufficient to prove certain patterns in these theories. However, no thought is given to decidability and completeness. We believe that the approach taken there is somewhat ad-hoc, and such a theory-spanning prover may be better architected with a decision procedure such as the one presented in this paper at its core.

3 Matching Logic Preliminaries

In this section, we formally introduce various matching logic preliminaries, including its formulae, semantics, models and theories.

Matching logic signatures prescribe the constant symbols and variables that may be used. Unlike in FOL, matching logic doesn't assign arities to symbols. In fact, all symbols are constants. More complex patterns, such as predicates and terms, may be built using the application construct shown later. Further, there are two different classes of variables — set variables, and elemental variables that are used in different contexts.

Definition 2. A matching logic *signature* $(\mathrm{EVar}, \mathrm{SVar}, \Sigma)$ has a set EVar of *element variables* $x, y, \ldots$, a set SVar of *set variables* $X, Y, \ldots$, and a set Σ of *(constant) symbols* $\sigma, f, g, \ldots$. We often omit EVar and SVar and use Σ to denote the signature.

Matching logic formalae are called *patterns*, and are build from structural constructs (constant symbols and applications) logical connectives, quantifiers and fixed-point constructs.

Definition 3. For a signature Σ, the set of matching logic *patterns*, denoted Pattern, is defined by the following grammar:

$$\varphi := \overbrace{\sigma \mid \varphi_1\ \varphi_2}^{\text{structural}} \mid \overbrace{\bot \mid \varphi_1 \to \varphi_2}^{\text{logical}} \mid \overbrace{x \mid \exists x.\, \varphi}^{\text{quantification}} \mid \overbrace{X \mid \mu X.\, \varphi}^{\text{fixpoints}}$$

where $\sigma \in \Sigma, x \in \mathrm{EVar}, X \in \mathrm{SVar}$, and X occurs positively in $\mu X.\varphi$ i.e., X is not nested in an odd number of times on the left of an implication $\varphi_1 \to \varphi_2$.

Pattern $\varphi_1\ \varphi_2$ is called *application* and assumed associative to the left. It may be used to build both terms and predicates. The scope of binders $\exists$ and μ goes farthest to the right. Observe that FOL-style quantifiers may only be used over element variables, whereas the fixedpoint operators only allow set variables. This disallows, for example, universal quantification over sets. The notions of free variables $\mathrm{FV}(\varphi)$, α-renaming, and capture-avoiding substitution ($\varphi[\psi/x]$ and $\varphi[\psi/X]$) are defined in the usual way. $\top, \bot, \wedge, \vee, \forall$ and ν are considered syntactic sugar and are defined as usual:

$$\begin{aligned}
\neg\varphi &\equiv \varphi \to \bot \\
\top &\equiv \neg\bot \\
\varphi_1 \vee \varphi_2 &\equiv \neg\varphi_1 \to \varphi_2 \\
\varphi_1 \wedge \varphi_2 &\equiv \neg(\neg\varphi_1 \vee \neg\varphi_2) \\
\forall x.\, \varphi &\equiv \neg\exists x.\, \neg\varphi \\
\nu X.\, \varphi &\equiv \neg\mu X.\, \neg\varphi[\neg X/X]
\end{aligned}$$

Intuitively, patterns have a "matching" semantics. For example, given list constructor symbols cons and nil, the pattern $\mathrm{cons}(x, \mathit{nil})$ matches all singleton lists. The patterns $\varphi_1 \wedge \varphi_2$ (resp. $\varphi_1 \vee \varphi_2$) match the elements in the model that match both φ_1 *and* φ_2 (resp. φ_1 *or* φ_2). Existentials have a union semantics, with $\exists x.\varphi(x)$ matching the union of all patterns that match $\varphi(a)$ for all elements a in the model. Similarly, universals have an intersection semantics.

The fixedpoint pattern $\mu X.\varphi(X)$ matches the least set A such that $\varphi(A) = A$. For example $\mu X.0 \vee s(s(X))$ matches the set of even natural numbers, when s is the successor function.

Definition 4. A matching logic *structure* is a three-tuple, $(M, \mathrm{app}_M, \{\sigma_M\}_{\sigma\in\Sigma})$, where :

- M is a nonempty, called *domain*;
- $\mathrm{app}_M \colon M \times M \to \mathcal{P}(M)$ is a binary *application* function;
- for each symbol σ, σ_M is a subset $\sigma_M \subseteq M$ called *interpretation* of $\sigma \in \Sigma$.

We use M to denote the above structure and also call it a *model*. We lift, *pointwise*, the application function app_M from over elements to over sets, as follows (where $A, B \subseteq M$)

$$\begin{aligned} \widetilde{\mathrm{app}}_M &\colon \mathcal{P}(M) \times \mathcal{P}(M) \to \mathcal{P}(M) \\ \widetilde{\mathrm{app}}_M(A, B) &= \bigcup_{a\in A, b\in B} \mathrm{app}_M(a, b) \end{aligned}$$

Note that $\widetilde{\mathrm{app}}_M(A, B) = \emptyset$ whenever $A = \emptyset$ or $B = \emptyset$. We overload the notation $\mathrm{app}_M(A, B)$ to mean $\widetilde{\mathrm{app}}_M(A, B)$ for simplicity.

Definition 5. Given $(\mathrm{EVar}, \mathrm{SVar}, \Sigma)$ and a structure M, a *valuation* is a function $\rho \colon (\mathrm{EVar} \cup \mathrm{SVar}) \to (M \cup \mathcal{P}(M))$ with $\rho(x) \in M$ for all $x \in \mathrm{EVar}$ and $\rho(X) \subseteq M$ for all $X \in \mathrm{SVar}$. Its *extension*, $|\_|_\rho$: Pattern $\to \mathcal{P}(M)$, is defined as:

$$\begin{aligned} |x|_\rho &= \{\rho(x)\} \text{ for } x \in \mathrm{EVar} \\ |X|_\rho &= \rho(X) \text{ for } X \in \mathrm{SVar} \\ |\sigma|_\rho &= \sigma_M \text{ for } \sigma \in \Sigma \\ |\varphi_1\varphi_2|_\rho &= \mathrm{app}_M(|\varphi_1|_\rho, |\varphi_2|_\rho) \\ |\bot|_\rho &= \emptyset \\ |\varphi_1 \to \varphi_2|_\rho &= M \setminus (|\varphi_1|_\rho \setminus |\varphi_2|_\rho) \\ |\exists x.\, \varphi|_\rho &= \bigcup_{a\in M} |\varphi|_{\rho[a/x]} \\ |\mu X.\, \varphi|_\rho &= \mu\mathcal{F} \text{ where } \mathcal{F}(A) = |\varphi|_{\rho[A/X]} \text{ for } A \subseteq M \end{aligned}$$

Here, "$\setminus$" is set difference; $\rho[a/x]$ (resp. $\rho[A/X]$) denotes the valuation ρ' such that $\rho'(x) = a$ (resp. $\rho'(X) = A$) and agrees with ρ on all other variables. $\mu\mathcal{F}$ denotes the least fixedpoint of $\mathcal{F}$, a monotone function. Its existence is guaranteed by the Knaster-Tarski fixpoint theorem [27]. We omit the subscript ρ when φ is closed is understood to be the empty valuation, and simply use $|\varphi|$.

Definition 6. We say pattern φ is *valid* in M, written $M \vDash \varphi$, iff $|\varphi| = M$ for all ρ : Var $\to M$. Let Γ be a set of Σ-patterns called *axioms*. We write $M \vDash \Gamma$ iff $M \vDash \psi$ for all $\psi \in \Gamma$. We write $\Gamma \vDash \varphi$ and say that φ is *valid* in Γ iff $M \vDash \varphi$ for all $M \vDash \Gamma$. We abbreviate $\emptyset \vDash \varphi$ as $\vDash \varphi$. We call the pair (Σ, Γ) a *matching logic Σ-theory*, or simply a *(Σ-)theory*. We say that M is *a model of the theory* (Σ, Γ) iff $M \vDash \Gamma$.

4 Status Quo

In this section we will review the existing decidability results for matching logic. Matching logic in its full generality is undecidable. Indeed, in [28], it is shown that matching logic can capture FOL as a theory. Since FOL is undecidable, matching logic must also be undecidable.

We should therefore consider subsets of matching logic instead. The obvious subsets to consider are syntactic fragments. We may consider fragments based on the components of the grammar defined in Definition 3 used to construct patterns and axioms. Let us first consider the fragment consisting only of the structural and logical components. That is, patterns that do not use the quantification and fixedpoints.

However, even with this basic fragment, we encounter a problem. The "word problem" for groups, a known undecidable problem [29], may be encoded as validity of patterns in this fragment, implying that satisfiability is undecidable. This problem is stated as:

> Given an arbitrary finitely presented group $\langle A \mid U_1 = \epsilon, \ldots, U_n = \epsilon \rangle$, where $A = \{a_1, a_2, \ldots a_n\}$, and $U_1, \ldots U_n$ are finite set of words in built from A and its inverses, are two words w_1 and w_2 equal?

Let $\Sigma = A \cup \bar{A} \cup \{\epsilon\}$, where $\bar{A} = \{a_1^{-1}, \ldots, a_n^{-1}\}$ represents the set of inverses, and ϵ represents the empty word Let $\mathrm{Eq} = \{U_1, \ldots, U_n\} \cup \{a_1 a_1^{-1}, \ldots, a_n a_n^{-1}\}$. Now, for each $e_1 e_2 \ldots e_n \in \mathrm{Eq}$ and each word $a_1 \ldots a_n \in A^*$ add a (countable) set of axioms $e_1(e_2(\ldots(e_n(a_1(a_2 \ldots (a_n)) \ldots)))) \leftrightarrow a_1(a_2 \ldots (a_n))$. Now, for two words $a_1 \ldots a_n$, $b_1 \ldots b_n$, the pattern $a_1\ (\ldots a_n) \leftrightarrow b_1\ (\ldots b_n)$ is valid iff they can be proved equal via the group equalities encoded by the axioms.

This encoding takes advantage of infinte axioms to emulate quantification, so this fragment may yet be decidable when the set of axioms if finite. However, a similar theory may be presented using only finitely-many axioms, if we allow either set variables or elemental variables in the axioms. Since even the simplest of fragment is undecidable for arbitrary theories, we must be more discerning regarding the theories we consider to obtain a non-trivial decidable fragment.

Let us, for now, restrict ourselves to decidability when $\Gamma = \emptyset$. Here, we find we can make more progress. If we restrict applications to only allow constant symbols in the first argument, we see that we are left with an embedding of modal logic, a decidable logic, in AML. We may even relax our restriction on the use of fixedpoint operators and have an embedding of modal μ-calculus. We shall in fact use this as the starting point for the results in this paper. In the next section we show that this restriction to constant symbols in first argument of applications is not neccessary for decidability. We begin to tackle non-empty theories in Section 6.

We summarize the status quo of the decidability of matching logic fragments in Table 7.

# of axioms	base	base	base	base, μ
in Γ	only	and μ	and $\exists$	and $\exists$
0	✓	$\boxed{\checkmark}$	✗	✗
Finite	?	✗	✗	✗
Infinite	✗	✗	✗	✗

Figure 7: Decidability results for syntactic fragments of matching logic with respect to cardinality of axioms in the theory.
Boxes denote nontrivial results proved in this paper.

$$\text{Base} \equiv \text{ structural} + \text{logical components}$$
$$\mu \equiv \text{fixedpoint component,}$$
$$\exists \equiv \text{quantification component}$$

5 Our Tableau-Based Decision Procedure

In this section, we prove the decidability of the empty theory for matching logic without existentials and set variables, that we will denote by $\mathrm{ML}^{\nexists\mu}$. At a high level, we present a two sets of tableau rules, S_{mod} and S_{ref}. If a tableau can be constructed via S_{mod}, and meets certain additional properties (in Definition 20), then the pattern under consideration must be satisfiable. Otherwise, a *refutation* may be constructed via the rules in S_{ref}. The tableau rules and proofs presented in this paper is inpired by a similar proof presented in [10] for modal μ-calculus, but extended to handle binary application with arbitrary patterns (whereas modal μ-calculus only allows binary application where the first argument is a constant symbol). In [10] a single set of rules is used to construct a tableau from which a model or refutation is obtained by pruning certain branches.

In the following subsections, we first define preliminaries for constructing tableau sequents, present the tableau rules, and then define conditions under which such a tableau is considered a "pre-model". Next, we prove a pattern is satisfiable iff a pre-model exists. We also show that a refutation may otherwise be constructed using S_{ref}, and that the time and space complexity of constructing the model is a function of the size of the formula, implying the small model property.

We only present outlines and intuitions for each of the proofs in the main body of the paper, and refer the reader to the appendix for complete versions.

5.1 Positive-Form Guarded Patterns

For technical convenience, we consider a more limited form of matching logic syntax without explicit negation, called the *positive form*:

Definition 9. Positive form patterns are patterns inductively defined according to the following grammar:

$$\begin{aligned}\varphi := {} & \overbrace{\sigma \mid \bar{\sigma} \mid \langle \varphi_1,\ \varphi_2 \rangle \mid [\varphi_1,\ \varphi_2]}^{\text{structural}} \\ & \mid \underbrace{\varphi_1 \wedge \varphi_2 \mid \varphi_1 \vee \varphi_2}_{\text{logical}} \mid \underbrace{X \mid \mu X.\, \varphi \mid \nu X.\, \varphi}_{\text{fixpoints}}\end{aligned}$$

where:

$$\begin{aligned}\bar{\sigma} &\equiv \neg\sigma & \langle \varphi_1,\ \varphi_2 \rangle &\equiv \varphi_1\ \varphi_2 \\ \nu X.\, \varphi &\equiv \neg \mu X.\, \neg\varphi[\neg X/X] & [\varphi_1,\ \varphi_2] &\equiv \neg\langle \neg\varphi_1,\ \neg\varphi_2 \rangle\end{aligned}$$

For the $\mathrm{ML}^{\nexists\mu}$fragment without axioms, this syntax is as expressive, but disallows negations of set variables. It is convenient, because we would otherwise have to handle each construct separately depending on whether it occured positively or negatively, does away with the need for the positivity condition for variables occuring in μ and ν. Every $\mathrm{ML}^{\nexists\mu}$pattern may be converted into a positive-form patten by applying De Morgan dualities and the equivalences in Definition 9.

For convenience, we show a derived semantics of positive-form patterns here. Given a model M and a valuation ρ, the semantics of positive-form patterns is:

$$\begin{aligned}|c| &= c_M \\ |\neg c| &= M \setminus c_M \\ |X| &= \rho(X) \\ |\varphi_1 \wedge \varphi_2| &= |\varphi_1| \cap |\varphi_2| \\ |\varphi_1 \vee \varphi_2| &= |\varphi_1| \cup |\varphi_2| \\ |\langle \varphi_1,\ \varphi_2 \rangle| &= \{a \in M \mid \exists a_1, a_2 \text{ s.t. } a \in \mathrm{app}_M(a_1, a_2) \\ &\qquad\qquad \wedge \forall i \in \{1,2\}, a_i \in |\varphi_i|\} \\ |[\varphi_1,\ \varphi_2]| &= \{a \in M \mid \forall a_1, a_2 \text{ s.t. } a \in \mathrm{app}_M(a_1, a_2) \\ &\qquad\qquad \to \exists i \in \{1,2\} \text{ s.t. } a_i \in |\varphi_i|\} \\ |\mu X.\, \varphi|_\rho &= \bigcap \Big\{ A \subseteq M \mid |\varphi|_{\rho[A/X]} \subseteq A \Big\} \\ |\nu X.\, \varphi|_\rho &= \bigcup \Big\{ A \subseteq M \mid A \subseteq |\varphi|_{\rho[A/X]} \Big\}\end{aligned}$$

Note that this presentation of the semantics is slightly different from that in Definition 5, describing μ (resp. ν) in terms of infinite union (resp. intersection) of the application of the function $\mathcal{F}$. Again, this is guaranteed to be the least (resp. greatest) fixed-point due to the Knaster-Tarksi theorem. We also present $\langle\ \rangle$ and $[\ ]$ to emphasize the symmetry between them. It also highlights that $\langle\ \rangle$ corresponds is the pointwise application of the app_M and allows $[\ ]$ to be presented showcasing its dual nature.

For technical reasons, we further restrict patterns to "guarded" patterns, that only allow bound set variables to occur within applications.

Definition 10 (Positive-form guarded pattern). Let φ be a positive-form pattern and X be a variable that occurs in φ (the occurrences may be free occurrences or bound occurrences). We say X is *guarded* in φ iff every occurrence of X is in scope of some $\langle\ \rangle$ or $[\,]$. We say a pattern is a *positive guarded pattern* iff it is a positive-form pattern and every bound variable is guarded.

Using some basic reasoning, we may prove that:

Lemma 11. *Every* $\mathrm{ML}^{\nexists\mu}$*pattern is equivalent to some positive-form guarded pattern.*

For example, given an unguarded pattern: $\mu X.\ \nu Y.\ X \vee \langle X,\ Y\rangle$, we may replace it with the equivalent pattern using the following derivation

$$\begin{aligned}
&\mu X.\ \nu Y.\ X \vee \langle X,\ Y\rangle \\
\Longleftrightarrow\ &\mu X.\ X \vee \langle X,\ \nu Y.\ X \vee \langle X,\ Y\rangle\rangle \\
\Longleftrightarrow\ &\mu X.\ \langle X,\ \nu Y.\ X \vee \langle X,\ Y\rangle\rangle
\end{aligned}$$

For the rest of this section, we will use assume that all patterns are positive-form guarded patterns.

5.2 Definition Lists

Definition 12. We extend the positive-form pattern syntax with a countable set, DefCons, of fresh constant symbols, called *definition constants*, denoted $U_1, U_2, \ldots, V_1, V_2, \ldots$. As with set variables, we only allow definition constants to appear *positively* in patterns, so they cannot have negations on top of them.

A *definition list* is a finite ordered list of equations:

$$\mathcal{D} = (U_1 = \kappa_1 X.\, \varphi_1)\,; \ldots ; (U_n = \kappa_n X.\, \varphi_n)$$

where $\kappa_1, \ldots, \kappa_n \in \{\mu, \nu\}$, $U_1, \ldots, U_n \in$ DefCons are distinct definition constants, X is a set variables, $\kappa_1 X.\, \varphi_1, \ldots, \kappa_n X.\, \varphi_n$ are distinct patterns (modulo α-equivalence), and for every $i \in \{1, \ldots, n\}$, all definition constants appearing in φ_i are among $U_1, \ldots, U_{i-1}$. If $i < j$, we say U_i is *older* than U_j and U_j is *younger* than U_i, with respect to the definition list $\mathcal{D}$. We call U_i a μ-constant or a ν-constant according to $\kappa_i = \mu$ or $\kappa_i = \nu$. We use ϵ to denote the empty definition list.

For every $i \in \{1, \ldots, n\}$, we call U_i the ith constant in $\mathcal{D}$ and $\kappa_i X.\, \varphi_i$ the *defining pattern* of U_i. Note that all defining patterns are fixpoint patterns that have the same binding variable X. This requirement is mainly for technical convenience and can be easily satisfied by α-renaming.

Definition 13. Given a pattern φ, we construct a *definition list for* φ by means of the contraction operation $\rrparenthesis \varphi \llparenthesis$ defined inductively as follows (where the operation $\circ$ is defined later):

1. $\rrparenthesis c \llparenthesis = \rrparenthesis \neg c \llparenthesis = \rrparenthesis X \llparenthesis = \rrparenthesis U \llparenthesis = \emptyset$;

2. $\rrparenthesis \varphi_1 \wedge \varphi_2 \llparenthesis = \rrparenthesis \varphi_1 \vee \varphi_2 \llparenthesis = \rrparenthesis \varphi_1 \llparenthesis \circ \rrparenthesis \varphi_2 \llparenthesis$;

3. $\rrparenthesis \langle \varphi_1,\ \varphi_2 \rangle \llparenthesis = \rrparenthesis [\varphi_1,\ \varphi_2] \llparenthesis = \rrparenthesis \varphi_1 \llparenthesis \circ \rrparenthesis \varphi_2 \llparenthesis$;

4. $\rrparenthesis \mu X.\, \varphi \llparenthesis = (U = \mu X.\, \varphi)\,;\, \rrparenthesis \varphi[U/X] \llparenthesis$, where U is a fresh definition constant;

5. $\rrparenthesis \nu X.\, \varphi \llparenthesis = (U = \nu X.\, \varphi)\,;\, \rrparenthesis \varphi[U/X] \llparenthesis$, where U is a fresh definition constant.

The operation $\rrparenthesis \varphi_1 \llparenthesis \circ \rrparenthesis \varphi_2 \llparenthesis$ is defined as follows. Firstly, we make sure that all the definition constants used in $\rrparenthesis \varphi_1 \llparenthesis$ are disjoint from those used in $\rrparenthesis \varphi_2 \llparenthesis$. Then, if there are $(U = \psi) \in \rrparenthesis \varphi_1 \llparenthesis$ and $(V = \psi) \in \rrparenthesis \varphi_2 \llparenthesis$ for the same pattern ψ, we delete $(V = \psi)$ from $\rrparenthesis \varphi_2 \llparenthesis$ and replace V with U in the remaining definitions in $\rrparenthesis \varphi_2 \llparenthesis$. Repeat this step until all the defining patterns are disjoint. Finally, let $\rrparenthesis \varphi_1 \llparenthesis \circ \rrparenthesis \varphi_2 \llparenthesis$ denote the sequence of the remaining definitions.

Intuitively, $\rrparenthesis \varphi \llparenthesis$ summarizes all the fixpoint sub-patterns of φ in a top-down way. The oldest (i.e., first) definition constant in $\rrparenthesis \varphi \llparenthesis$ corresponds to the topmost fixpoint sub-pattern of φ, say, $\mu X_1.\, \varphi_1$. Then, the fixpoint sub-patterns of $\varphi_1[U_1/X_1]$ are recursively summarized, where all the (recursive) occurrences of X_1 are replaced with U_1. Therefore, for every $(U_i = \varphi_i) \in \rrparenthesis \varphi \llparenthesis$, the subscript i denotes the *timestamp* that shows the time when φ_i is summarized. If U_i is *older* than U_j, then the defining pattern U_i is summarized *before* that of U_j is summarized. Since $\rrparenthesis \varphi \llparenthesis$ is computed in a top-down way, $U_1, \ldots, U_n$ are listed in the pre-order traversal of the parse tree of φ. Let us demonstrate this operator:

$$\begin{aligned}
& \rrparenthesis (\nu Y. p \wedge \langle \bullet,\ Y \rangle) \llparenthesis \\
\equiv\ & (V_1 = \nu Y. p \wedge \langle \bullet,\ Y \rangle); \rrparenthesis p \wedge \langle \bullet,\ V_1 \rangle \llparenthesis \\
\equiv\ & (V_1 = \nu Y. p \wedge \langle \bullet,\ Y \rangle); \rrparenthesis p \llparenthesis \circ \rrparenthesis \langle \bullet,\ V_1 \rangle \llparenthesis \\
\equiv\ & (V_1 = \nu Y. p \wedge \langle \bullet,\ Y \rangle); \rrparenthesis \langle \bullet,\ V_1 \rangle \llparenthesis \\
\equiv\ & (V_1 = \nu Y. p \wedge \langle \bullet,\ Y \rangle); \rrparenthesis \bullet \llparenthesis \circ \rrparenthesis V_1 \llparenthesis \\
\equiv\ & (V_1 = \nu Y. p \wedge \langle \bullet,\ Y \rangle)
\end{aligned}$$

Definition 14. Let φ be a pattern that may contain definition constants and $\mathcal{D}$ be a definition list that contains all definition constants appearing in φ. We define the *expansion operation* $\llparenthesis \varphi \rrparenthesis_{\mathcal{D}}$ that subsequently replaces definition constants in φ by their defining patterns according to $\mathcal{D}$, as:

$$\llparenthesis \varphi \rrparenthesis_{\mathcal{D}} = \varphi[\varphi_n/U_n] \cdots [\varphi_1/U_1]$$

where $\mathcal{D} = (U_1 = \varphi_1), \ldots, (U_n = \varphi_n)$

Note that expansion is carried out from the youngest definition constants to the oldest, in reserve to the contraction operation $\rrparenthesis \varphi \llparenthesis$. When $\mathcal{D}$ is understood, we abbreviate $\llparenthesis \varphi \rrparenthesis_{\mathcal{D}}$ as $\llparenthesis \varphi \rrparenthesis$. For example, $\llparenthesis V_1 \rrparenthesis_{\mathcal{D}} \equiv \nu Y. p \wedge \langle \circ,\ Y \rangle$, where $\mathcal{D}$ is the definition list from above.

5.3 Tableau Sequents

Below, we define various constructs needed for tableau sequents.

Definition 15. Let Γ be a set of patterns. We define $\Gamma_{\langle\,\rangle}$ (resp. $\Gamma_{[\,]}$) to be the set of $\langle\,\rangle$-patterns (resp. $[\,]$-patterns) in Γ. Formally,

$$\Gamma_{\langle\,\rangle} = \{\langle\varphi_1,\ \varphi_2\rangle \mid \langle\varphi_1,\ \varphi_2\rangle \in \Gamma\}$$
$$\Gamma_{[\,]} = \{\,[\varphi_1,\ \varphi_2]\ \mid\ [\varphi_1,\ \varphi_2] \in \Gamma\}$$

Definition 16. Given Γ we define a *witness function* $\mathrm{wit}\colon \Gamma_{[\,]} \to \{1,2\}$ as a function that maps every pattern of the form $[\psi_1,\ \psi_2] \in \Gamma$ to either 1 or 2, called the *witness*. Let $\mathrm{Wit}(\Gamma) = [\Gamma_{[\,]} \to \{1,2\}]$ denote the set of all witness functions with respect to Γ. For a witness function wit, let $\Gamma_i^{\mathrm{wit}} = \{\psi_i \mid [\psi_1,\ \psi_2] \in \Gamma$ and $\mathrm{wit}([\psi_1,\ \psi_2]) = i\ \}$.

Each witness function represents one possible partition of $[\,]$ terms into two sets. For example, when $\Gamma_{[\,]} = \{[a_1,\ a_2], [b_1,\ n_2]\}$ there are four witness possible functions:

$$a = \{[a_1,\ a_2] \mapsto 1, [b_1,\ b_2] \mapsto 1\}$$
$$b = \{[a_1,\ a_2] \mapsto 1, [b_1,\ b_2] \mapsto 2\}$$
$$c = \{[a_1,\ a_2] \mapsto 2, [b_1,\ b_2] \mapsto 1\}$$
$$d = \{[a_1,\ a_2] \mapsto 2, [b_1,\ b_2] \mapsto 2\}$$

Choosing witness c, we have $\Gamma_1^c = \{b_1\}$ and $\Gamma_2^c = \{a_2\}$

Definition 17. A *tableau sequent* is one of the following:

1. a finite nonempty pattern set Γ;
2. $\Gamma \rightsquigarrow \langle\varphi_1,\ \varphi_2\rangle$, where $\langle\varphi_1,\ \varphi_2\rangle \in \Gamma$;
3. $\Gamma \rightsquigarrow \langle\varphi_1,\ \varphi_2\rangle \rightsquigarrow \mathrm{wit}$, where $\langle\varphi_1,\ \varphi_2\rangle \in \Gamma$ and $\mathrm{wit} \in \mathrm{Wit}(\Gamma, \sigma)$.

We call (1) a *normal sequent* and (2) and (3) *ghost sequents*.

Definition 18. For a normal sequent Γ, we say Γ is an *inconsistent sequent* if there exists $\varphi \in \Gamma$ such that its negation is in Γ. If a normal sequent is not inconsistent, then it is a consistent sequent.

5.4 Constructing a Tableau

Let $\mathcal{D}$ be a definition list. We define the following set S_{mod} of *tableau rules* with respect to $\mathcal{D}$ as shown in 8b, where Γ is a finite nonempty set of sentences and $\varphi, \varphi_1, \varphi_2$ are sentences, whose definition constants are all contained in $\mathcal{D}$.

Definition 19. A *tableau* for ψ is a possibly infinite labeled tree (T, L), where T is a tree whose set of nodes is $\mathrm{Nodes}(T)$ and nodes are denoted $s, s_1, s_2, \ldots$, and the root node is $\mathrm{root}(T)$. The labeling function $L\colon \mathrm{Node} \to \mathrm{Sequent}$ associates every node of T with a sequent, such that the following conditions are satisfied:

1. $L(\mathrm{root}(T)) = \{\psi\}$;

2. For every $s \in \mathrm{Nodes}(T)$, if $L(s)$ is an inconsistent sequent then s is a leaf of T;

3. For every $s \in \mathrm{Nodes}(T)$, if $L(s)$ is not an inconsistent sequent and one of the tableau rules in S_{mod} can be applied (with respect to the definition list $\mathcal{D} = \rrbracket\psi\llbracket$), and the resulting sequents are $\mathrm{seq}_1, \dots, \mathrm{seq}_k$, then s has exactly k child nodes $s_1, \dots, s_k$, and $L(s_1) = \mathrm{seq}_1$, $\dots$, $L(s_k) = \mathrm{seq}_k$.

In (3), we categorize the nodes by the corresponding tableau rules that are applied. For example, if the a child nodes of s is obtained by applying (OR-L), then we call s an (OR-L) node. We also categorize the nodes by their labeling sequents. If a node is labeled with a normal sequent, we call it a normal node. Otherwise, it is labeled with a ghost sequent and we call it a ghost node. Note that a node is a ghost node iff it is an (APP$_2$) or (APP$_3$) node. In either case, its closest ancestor normal node is an (APP$_1$) node.

For any tableau (T, L), the leaves of T are either labeled with inconsistent sequents, or they are (APP$_1$) nodes whose labels contain no σ-patterns for any σ. For any non-leaf node, unless it is labeled with (APP$_1$) or (APP$_3$), it has exactly one child node.

Definition 20. A *quasi-model* is a tableau where all leaf nodes are consistent.

Thus, in a quasi-model, all leafs are (APP$_1$) nodes.

Definition 21. Let ψ be a sentence and (T, L) be a tableau for ψ. Given a rooted maximal (possibly infinite) path P of T, a *trace* on P is a partial function $\mathrm{Tr}\colon P \mapsto \mathrm{Pattern}$ whose domain $\mathrm{dom}(\mathrm{Tr})$ is a prefix of P, such that the following conditions are satisfied:

1. If $\mathrm{Tr}(s)$ is defined on $s \in \mathrm{Nodes}(T)$, and

   (a) if s is a normal node then $\mathrm{Tr}(s) \in L(s)$;

   (b) if s is a ghost node, then let s' be the closest ancestor normal node of n and define $\mathrm{Tr}(s) = \mathrm{Tr}(s')$;

2. $\mathrm{Tr}(\mathrm{root}(T))$ is defined, and by (1), $\mathrm{Tr}(\mathrm{root}(T)) = \psi$;

3. If $\mathrm{Tr}(s)$ is defined on $s \in \mathrm{Nodes}(T)$ and s' is the next node of s in P that is a normal node and is obtained *not* by applying (APP$_3$), and

   (a) if the rule does not reduce $\mathrm{Tr}(s)$, then we define $\mathrm{Tr}(s') = \mathrm{Tr}(s)$;

   (b) if the rule reduces $\mathrm{Tr}(s)$, then we let $\mathrm{Tr}(s')$ be one of the results of the reduction, non-deterministically. Note that the nondeterministic choice only occurs when s is an (AND) node, $L(s) = \{\varphi_1 \wedge \varphi_2\} \cup \Gamma$, $T(s) = \varphi_1 \wedge \varphi_2$, and $L(s') = \{\varphi_1, \varphi_2\} \cup \Gamma$. In this case, $T(s')$ is φ_1 or φ_2, non-deterministically.

4. If $\mathrm{Tr}(s)$ is defined on $s \in \mathrm{Nodes}(T)$ whose label $L(s) = \Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle \rightsquigarrow$ wit, and s' is the next node of s in P obtained by applying ($\textsc{app}_3$), and $L(s') = \{\varphi_i\} \cup \Gamma_i^{\mathrm{wit}}$ for some $i \in \{1, 2\}$, and
   (a) if $\mathrm{Tr}(s) = \langle \varphi_1,\ \varphi_2 \rangle$, then $\mathrm{Tr}(s') = \varphi_i$;
   (b) if $\mathrm{Tr}(s) = [\psi_1,\ \psi_2]$ and $\psi_i \in \Gamma_i^{\mathrm{wit}}$, then $\mathrm{Tr}(s') = \psi_i$;
   (c) $\mathrm{Tr}(s')$ is undefined for any other cases.

Definition 22. We say a definition constant U *regenerates* on Tr if exists a node s such that $\mathrm{Tr}(s) = U$ and $\mathrm{Tr}(s') = \kappa X.\, \varphi[U/X]$, where s' is the next node of n on Tr and $(U = \kappa X.\varphi) \in \mathcal{D}$. We say Tr is a κ-trace for $\kappa \in \{\mu, \nu\}$, if it is infinite and the oldest definition constant (with respect to $\mathcal{D}$) that regenerates infinitely often is a κ-constant.

Lemma 23. *Any infinite trace is either a μ-trace or a ν-trace.*

Definition 24. A quasi-model is called a *pre-model* iff all infinite traces on all paths are ν-traces.

5.5 Correspondence between existance of pre-models and satisfiability

Now that we have know how to build pre-models, we go about proving decidability:

Theorem 25. *For any positive guarded sentence ψ, determining whether ψ is satisfiable is decidable.*

To do this, we prove the equivalence between pre-models and satisfiability.

Theorem 26. *For any positive guarded sentence ψ, there exists a pre-model for ψ iff ψ is satisfiable.*

We handle each direction separately in the following propositions:

Proposition 27. *If a positive guarded sentence ψ is satisfied in M on a, there exists a pre-model for ψ.*

and

Proposition 28. *If there exists a pre-model for a positive guarded sentence ψ then ψ is satisfiable.*

Proposition 27 is proved by constructing a pre-model from a model M of ψ and an element $a \in |\psi|$. As we construct each node, we assign an element a_s to the node and maintain the invariant $L(s) \models a_s$. The invariant allows us to construct further nodes, resulting in a quasi-nodel. We then show that the quasi-model must be a pre-model, or else there would be a μ-trace.

Next, to prove Proposition 28 we construct a model, called the canonical model, from the pre-model. We define the set of elements in the model as the $(\textsc{app}_1)$ nodes (including the leaf nodes with zero $\langle\ \rangle$-patterns. For nodes $s, s_1, s_2 \in M$, $\mathrm{app}_M(s_1, s_2) = s$ iff s_1 and s_2's nearest $(\textsc{app}_1)$ descendant is s.

While most of the tableau rules have an intuitive meaning, replacing the current sequent with a set of equisatisfiable sequents, the $(\textsc{app}_i)$ rules need some explanation. Let us review the semantics of $\langle\ \rangle$ and $[\ ]$.

$$\begin{aligned}
|\langle\varphi_1,\ \varphi_2\rangle| &= \{a \in M \mid \exists a_1, a_2 \text{ s.t. } a \in \mathrm{app}_M(a_1, a_2) \\
&\qquad\qquad\qquad \wedge \forall i \in \{1, \dots, n\}, a_i \in |\varphi_i|\} \\
|[\varphi_1,\ \varphi_2]| &= \{a \in M \mid \forall a_1, a_2 \text{ s.t. } a \in \mathrm{app}_M(a_1, a_2) \\
&\qquad\qquad\qquad \to \exists i \in \{1, 2\} \text{ s.t. } a_i \in |\varphi_i|\}
\end{aligned}$$

Each instance of $\langle\varphi_1,\ \varphi_2\rangle$ requires that there exists some elements a_1 and a_2, that are matched by φ_1 and φ_2 and that $a \in \mathrm{app}_M(a_1, a_2)$. But now, each instance of $[\varphi_1,\ \varphi_2]$ we require that for every element in $\mathrm{app}_M(a_1, a_2)$, there is some $i \in \{1, 2\}$, a_i matches φ_i. So, for each $[\varphi_1,\ \varphi_2]$ the either s_1 must match φ_1, or s_2 must match φ_2. Thus, we need to assign each $[\ ]$ to either s_1, or s_2. This is precisely what the witness does. So, intuitively, each $(\textsc{app}_1)$ rule selects all $\langle\ \rangle$ patterns, the $(\textsc{app}_2)$ node chooses a witness, and $(\textsc{app}_3)$ begins construction of the nodes.

5.6 Refutations

We may use the rules in S_{ref} to build a refutation as defined below.

Definition 30. A *refutation* for a pattern ψ is a possibly infinite labeled tree (T, L), where T is a tree whose set of nodes is $\mathrm{Nodes}(T)$ and nodes are denoted $s, s_1, s_2, \dots$, and the root node is $\mathrm{root}(T)$. The labeling function $L\colon \mathrm{Node} \to \mathrm{Sequent}$ associates every node of T with a sequent, such that the following conditions are satisfied:

1. $L(\mathrm{root}(T)) = \{\psi\}$;

2. For every $s \in \mathrm{Nodes}(T)$, if $L(s)$ is an inconsistent sequent then s is a leaf of T;

3. For every $s \in \mathrm{Nodes}(T)$, if $L(s)$ is not an inconsistent sequent and one of the tableau rules in S_{ref} can be applied (with respect to the definition list $\mathcal{D} = \rrparenthesis\psi\llparenthesis$), and the resulting sequents are $\mathrm{seq}_1, \dots, \mathrm{seq}_k$, then s has exactly k child nodes $s_1, \dots, s_k$, and $L(s_1) = \mathrm{seq}_1, \dots, L(s_k) = \mathrm{seq}_k$.

4. *Every* leaf node is labeled with an inconsistent sequent.

5. *Every* infinite path has μ-trace.

We may prove that:

Theorem 31. *For an arbitary positive-form guarded pattern φ, there exists either a pre-model or a refutation.*

This may be deduced by constructing a Gale and Stewart game[30] with Borel winning[31] conditions. One player attempts to find a model, while the other, a refutation. Borel winning conditions imply that there must be a winner, i.e. that there must be either a model, or a refutation.

5.7 Example: Modal μ-Calculus Tableaux as an Instance of the Matching Logic Tableau

We begin with a review of the syntax of modal μ-calculus. Note that here we present the version with muliple modalities, there also exists a simpler version with just a single modalilty. Given a finite set of atomic propositions P, a finite set of actions A, and a countable of variables V, the language of Modal μ-calculus formula is defined inductively from the following grammar:

$$\varphi := p \mid X \mid \varphi \wedge \varphi \mid \neg\varphi \mid \langle a\rangle\varphi \mid \mu X.\varphi$$

where $p \in P$, $a \in A$, and $X \in V$, and X occurs positively in φ. Derived operators are defined as sugar: $[a]\varphi \equiv \langle a\rangle\neg\varphi$ and $\nu X.\varphi \equiv \neg\mu X.\neg\varphi[X/\neg X]$.

This is translated into an matching logic theory whose signature consists of a symbol for each $p \in P$, for each $a \in A$. The modal operator is translated to application via the alias $\langle\varphi_1\rangle\varphi_2 \equiv \langle\varphi_1,\ \varphi_2\rangle$. From this we may derive that $[a]\varphi \equiv [\neg a,\ \varphi]$. Modal μ-calculus formalae as represented in matching logic thus only have constants as the first argument of $\langle\ \rangle$, and negations of constants as that of $[\ ]$.

[10] defines a set of tableau rules identically to those defined here, except that the $(\textsc{app}_i)$ rules are replaced with a single rule:

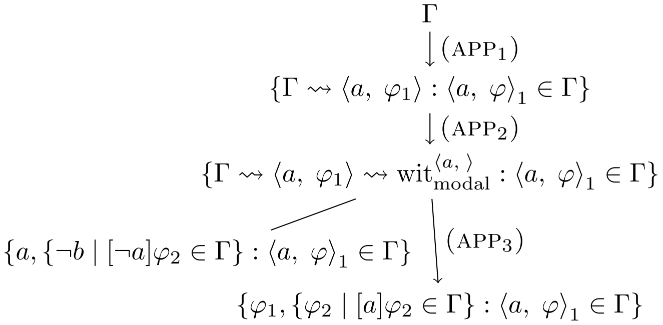
$$\frac{\Gamma}{\{\alpha, \{\beta : [a]\beta \in \Gamma\} : \langle a\rangle\alpha \in \Gamma\}}\ (\textsc{all}\langle\rangle)$$

We show that every mu-calculus tableau may be considered an instance of a matching logic tableau.

Let us consider the case when the sequent contains $a = \langle a\rangle\varphi_1 \equiv \langle a,\ \varphi_1\rangle$ and $\bar{a} = [a]\psi_2 \equiv [\neg a,\ \varphi_2]$. Consider any witness where $\mathrm{wit}(\bar{a}) = 1$. Attempting to apply the $(\textsc{app}_i)$ rules will result in an inconsistent node, as shown in Figure 32a. To produce a model, we must therefore always use a witness where $\mathrm{wit}(s) = 2$ when the first argument of mu calculus' $\langle\ \rangle$ and $[\ ]$ are the same constant symbol.

Next, consider the case when the sequent contains $a = \langle a\rangle\varphi_1 \equiv \langle a,\ \varphi_1\rangle$ and $\bar{b} = [b]\psi_2 \equiv [\neg b,\ \varphi_2]$ (see Figure 32b). Choosing a witness when $\mathrm{wit}(\bar{b}) = 1$ results in the first child of $(\textsc{app}_3)$ with only the constant a and the negations of *other* constants. Since $\neg b$ only conflicts with b, this is a safe choice. We do not need to consider φ_2 on the second child of $(\textsc{app}_3)$. All other models would put additional constraints on the right child of the $(\textsc{app}_3)$ node.

Defining $\mathrm{wit}^{\langle a,\ \rangle}_{\mathrm{modal}}$ as assigning $[\neg a,\ \varphi_1]$ to 1, and $[\neg b,\ \varphi_1]$ to 2 when $a \neq b$, we get:


Γ
(APP_1)
$\{\Gamma \rightsquigarrow \langle a,\ \varphi_1\rangle : \langle a,\ \varphi\rangle_1 \in \Gamma\}$
(APP_2)
$\{\Gamma \rightsquigarrow \langle a,\ \varphi_1\rangle \rightsquigarrow \text{wit}^{\langle a,\ \rangle}_{\text{modal}} : \langle a,\ \varphi\rangle_1 \in \Gamma\}$
$\{a, \{\neg b \mid [\neg a]\varphi_2 \in \Gamma\} : \langle a,\ \varphi\rangle_1 \in \Gamma\}$
(APP_3)
$\{\varphi_1, \{\varphi_2 \mid [a]\varphi_2 \in \Gamma\} : \langle a,\ \varphi\rangle_1 \in \Gamma\}$

5.8 Example: Dynamic logic

Dynamic logic is a common logic used in program reasoning. It extends modal logic by allowing more complex modalities, such as sequences of actions, loops, choices and conditionals, described by the grammar:

$$\begin{aligned}\varphi &:= p \mid \varphi \to \varphi \mid \text{false} \mid [\alpha]\varphi \\ \alpha &:= a \mid \alpha\,;\alpha \mid \alpha \cup \alpha \mid \alpha^* \mid \varphi?\end{aligned}$$

As with modal μ-calculus, dynamic logic may also be translated to the $\text{ML}^{\nexists\mu}$ fragment, without any axioms:

$$\begin{array}{ll} \langle\alpha\rangle\varphi \equiv (\bullet\ \alpha\ \varphi) & [\alpha]\varphi \equiv \neg\langle\alpha\rangle\neg\varphi \\ (\textsc{Seq})\ [\alpha\,;\beta]\varphi \equiv [\alpha][\beta]\varphi & (\textsc{Choice})\ [\alpha\cup\beta]\varphi \equiv [\alpha]\varphi \wedge [\beta]\varphi \\ (\textsc{Test})\ [\psi?]\varphi \equiv (\psi \to \varphi) & (\textsc{Iter})\ [\alpha^*]\varphi \equiv \nu X.\,(\varphi \wedge [\alpha]X) \end{array}$$

Notice that this translation takes advantage of matching logic's more powerful application, using it to build a nested application for the translation of $\langle\alpha\rangle\varphi$. Figure 33 shows an example tableau for a dynamic logic formula.

6 Linear Temporal Logic: Extending to non-empty theories

In this section, we take a look at how we may expand the decidable fragment beyond empty theories, and begin to handle theories with axioms. We use linear temporal logic as a case-study, because it is defined by a simple theory.

For the signature, we use the set of propositional constants and a symbol $\circ$. LTL is defined as a matching logic theory using a small set of notations and two axioms:

$$\text{“weak next”} \qquad\qquad \bullet\varphi \equiv \neg(\circ\neg\varphi)$$

$$\begin{array}{rc}
\text{“eventually”} & \Diamond\varphi \equiv \mu X.\ \varphi \vee \bullet X \\
\text{“always”} & \Box\varphi \equiv \nu X.\ \varphi \wedge \circ X \\
\text{“(strong) until”} & \varphi_1\ U\ \varphi_2 \equiv \mu X.\ \varphi_2 \vee (\varphi_1 \wedge \bullet X)
\end{array}$$

$$(\textsc{Inf})\ \bullet\top \qquad\qquad (\textsc{Lin})\ \bullet X \to \circ X$$

The two axioms (Lin) and (Inf) force linear infinite models, and can be used to derive an equivalent axiom $\bullet X \leftrightarrow \circ X$. That is, that $\circ$ and its dual $\bullet$ are equivalent. With this insight, we modify the tableau S_{mod} to require that a new proof rule is applied before $(\textsc{app}_1)$:

$$\frac{\langle\circ,\ \Delta_1\rangle, [\neg\circ,\ \Delta_2], \Gamma}{\langle\circ,\ \Delta_1\rangle, \langle\circ,\ \Gamma_2\rangle, [\neg\circ,\ \Delta_1], [\neg\circ,\ \Delta_2], \Gamma}\ (\textsc{equiv})$$

when † holds and Γ does not contain any $[\,]$ and $\langle\,\rangle$ patterns.

Here, $\langle\circ,\ \Delta_1\rangle$ (resp $[\neg\circ,\ \Delta_2]$) represents the *set* of all $\langle\,\rangle$ patterns (resp. $[\,]$) with $\circ$ (resp. $\neg\circ$) as their first argument.

While this change by itself does not produce linear models, linear models may be derived from these models. To see why this is the case, we take a look at the sequents produced by applying the (equiv), $(\textsc{app}_1)$, $(\textsc{app}_2)$ and $(\textsc{app}_3)$ rules.

$$\begin{tikzcd}
 & \langle\circ,\ \Delta_1\rangle, [\neg\circ,\ \Delta_1], \Gamma(0) \arrow[d, "(\textsc{equiv})"] & \\
 & \langle\circ,\ \Delta_1 \cup \Delta_2\rangle, [\neg\circ,\ \Delta_1 \cup \Delta_2] \arrow[d, "(\textsc{app}_1)"] & \\
 & \{\langle\circ,\ \varphi_1\rangle \rightsquigarrow [\circ, \Delta_1 \cup \Delta_2,\ ,][\neg\circ,\ \Delta_1 \cup \Delta_2] \mid \text{for } \varphi_1 \in \Delta_1 \cup \Delta_2\} \arrow[d, "(\textsc{app}_2)"] & \\
 & \{\langle\circ,\ \varphi_1\rangle \rightsquigarrow \text{wit} \rightsquigarrow \langle\circ,\ \Delta_1 \cup \Delta_2\rangle, [\neg\circ,\ \Delta_1 \cup \Delta_2] \mid \text{for } \varphi_1 \in \Delta_1 \cup \Delta_2\} \arrow[dl, dash] \arrow[dr, "(\textsc{app}_3)"] & \\
\{\circ \mid \text{for } \varphi_1 \in \Delta_1 \cup \Delta_2\} & & \{\Delta_1 \cup \Delta_2(2) \mid \text{for } \varphi_1 \in \Delta_1 \cup \Delta_2\}
\end{tikzcd}$$

First, notice that as with the case of model μ-calculus, only one choice of witness can produce models, $\text{wit}^{\langle a,\ \rangle}_{\text{modal}}$, defined in the previous section as assigning each $[\,]$ term to 2. Next, observe that, due to the application of the (equiv), while there are several sequents produced after the application of the $(\textsc{app}_3)$ rule, only two are unique. The first set of children are labeled with a single pattern $\circ$ and is a leaf node. The second set of sequents are all labeled with the right sub-patterns of the $\langle\,\rangle$ and $[\,]$ patterns in the original sequent — i.e. they all have the same label. For any pre-model, we may build a linear LTL model by ignoring the first set of children, and treating the second set of sequents as an equivalence class, choosing the children from any arbitrary node in the set.

Another important insight is that whenever $\alpha\, U\, \beta$ (or $\Diamond\beta$) occurs as the label of a node, a future state must eventually be labeled with β. This is because all traces in the pre-model are ν-traces. If a trace is labeled $\alpha\, U\, \beta \equiv \mu X.\beta \vee (\alpha \wedge \bullet X)$ and no future state has the label β, then we must choose the (OR-R) rule infinitely often. But then, the corresponding definitional μ-constant is regenerated infinitely often, without any older ν-constant between regenerations. Thus it must have a μ-trace, a contradiction.

With these, we may prove that for the modified tableau rules:

Theorem 35. *Every pre-model corresponds to an LTL model.*

Figure 34 shows some example tableaux for LTL formulae.

7 Implementation

We've implemented the matching logic tableau as a rewriting system using Maude [32]. The rewriting system has as its state, sets of annotated sequents. Since the representational distance between tableaux/proof-trees and rewriting is low, For the most part, the implementation of each tableaux rule looks almost identical to the tableaux rules. There is, however, some bookkeeping we must do to facilitate searching for models, and to detect μ- and ν-traces.

There are two things the bookkeeping needs to handle. First, there are a few sequents that may have multiple possible children. This happens when (OR-L) and (OR-R), (APP$_1$), (APP$_2$) or (APP$_3$) may apply. However, when (OR-L) and (OR-R), and (APP$_2$) may apply we need to choose any one of the possible children, whereas when (APP$_1$) and (APP$_3$) may apply, we must choose all children.

We represent the choice of possible child sequents as non-deterministic rules. For example, the maude rules for (OR-L) and (OR-R)

```
crl [or-l] : ⟨ \or(P1, P2), Gamma ; DefList ; Hist ⟩
          ⇒ ----------------------------------------
             [ P1, Gamma ; DefList ; Hist ]
    if P1 ≠ ⊥ ∧ P2 ≠ ⊥ .

crl [or-r] : ⟨ \or(P1, P2), Gamma ; DefList ; Hist ⟩
          ⇒ ----------------------------------------
             [ P2, Gamma ; DefList ; Hist ]
    if P1 ≠ ⊥ ∧ P2 ≠ ⊥ .
```

Similarly, the (APP$_2$) rule partitions each of the [ ] patterns into two sets, corresponding to an assignment of witness via the following non-deterministic rules.

```
rl ⟨ ... ⇝ wit {P1s}{P2s} ⇝ [DApp1 DApp2], Gamma; ... ⟩
⇒ -------------------------------------------------------
   ⟨ ... ⇝ wit {P1s,DApp1}{P2s} ⇝ Gamma; ... ⟩ .

rl ⟨ ... ⇝ wit {P1s}{P2s} ⇝ [DApp1 DApp2], Gamma; ... ⟩
⇒ -------------------------------------------------------
   ⟨ ... ⇝ wit {P1s}{P2s,DApp2} ⇝ Gamma; ... ⟩ .
```

Handling multiple child sequents, such as with the (APP$_1$) and (APP$_3$) rules, is done by sequent conjunction operator, &&.

```
eq ⟨ ⟨ P1 P2 ⟩ ⇝ wit { P1s } { P2s } ⇝ empty; ...⟩
 = ---------------------------------------------------
   ⟨ P1, P1s ;...⟩ && ⟨ P2, P2s; ...⟩
```

The second piece of bookkeeping we need to perform is the detection of μ -and ν-traces. There are two parts to doing this. Firstly, we must detect when a path loops back on itself so that we may stop applying tableau rules on this branch. We do this be accumulating a history of all the sequents in that path, and when a sequent repeats, we stop execution. Secondly, we must check whether we have a $\mu-$ trace. We do this by annotating each possible trace-label in the sequent with the set of definitional variables that have previously been generated. When a sequent recurrs we check if the oldest constant generated between occurrences on each trace is a μ- or a ν-constant. If it is a μ-constant, we fail that branch. Otherwise, we consider the branch successful.

8 Future work

8.1 Extension to Equational Theories

The modified LTL tableau presented in Section 6 could in principle be extended to other theories with similar axioms — equations whose application results in the patterns labeling the sequent reaching a fixedpoint. Besides LTL, such theories could include algebraic and co-algebraic structures. For each pair of distinct constructors (without axioms) f, g, we may derive the following propositions: $\langle f,\ \varphi\rangle \to [\neg g,\ \bot]$ that forces models matching $f(x)$ to not match $g(y)$; and $\langle f,\ \varphi\rangle \to [\neg f,\ \varphi]$ that forces injectivitiy of constructors. Note that these axioms alone allow both inductive and co-inductive structures. While is it clear that these axioms are sound (i.e. that they follow from the properties of constructors), it is not clear that they are enough to rule out all models where constructor axioms do not hold. The (EQUIV) may be modified to take this implications into account instead of the ones mentioned in Section 6 to allow working with there theories. A similar rule for associative and associative-commutative constructors would require a relaxation of these axioms.

8.2 Guarded Fragment of FOL+fixedpoints

In [20], it is claimed that the reason for the robustness of the decidability of modal logic is that it translates to the "guarded fragment" of FOL. This guarded fragment, even with fixedpoint operators introduced, is decidable. It is therefore worth looking into how this fits into a decision procedure for a fragment of matching logic.

8.3 As the basis for an SMT prover for matching logic

For FOL, most modern satisfiability modulo theories (SMT) provers use a decision procedure as there core. Many work by separating the formula into a predicate logic "skeleton" and a first-order part. This skeleton is solved using the DPLL [33] decision procedure or similar. The SMT solver then iterates through each of the models returned by this procedure, and checks if they are "compatible" with the first-order part. A similar architecture may prove useful for an matching logic solver. Since $\mathrm{ML}^{\nexists\mu}$is a larger, decidable fragment of AML than propositional logic, it may prove a better basis for such solver.

9 Conclusion

In this work, we proved the decidability of the first non-trivial fragment of matching logic. This lays a foundation which we may extend to identify additional decidable fragments of matching logic. We may also use this decision procedure as the basis of a matching logic solver.

References

[1] G. Roşu, "Matching logic," *Logical Methods in Computer Science*, vol. 13, no. 4, pp. 1–61, Dec. 2017.

[2] X. Chen and G. Roşu, "Applicative matching logic: Semantics of K," University of Illinois at Urbana-Champaign, Tech. Rep. http://hdl.handle.net/2142/104616, July 2019.

[3] The K Team. The K framework. [Online]. Available: https://kframework.org/

[4] X. Chen and G. Roşu, "Applicative matching logic," University of Illinois at Urbana-Champaign, Tech. Rep. http://hdl.handle.net/2142/104616, 2019.

[5] X. Chen and G. Roşu, "Matching μ-logic," in *2019 34th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*. IEEE, 2019, pp. 1–13.

[6] G. Roşu, "Matching logic," *arXiv preprint arXiv:1705.06312*, 2017.

[7] X. Chen, D. Lucanu, and G. Roşu, "Connecting constrained constructor patterns and matching logic," in *International Workshop on Rewriting Logic and its Applications*. Springer, 2020, pp. 19–37.

[8] X. Chen and G. Roşu, "A general approach to define binders using matching logic," *Proceedings of the ACM on Programming Languages*, vol. 4, no. ICFP, pp. 1–32, 2020.

[9] M. d'Agostino, "Tableau methods for classical propositional logic," in *Handbook of tableau methods*. Springer, 1999, pp. 45–123.

[10] D. Niwiński and I. Walukiewicz, "Games for the μ-calculus," *Theoretical Computer Science*, vol. 163, no. 1, pp. 99–116, 1996. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0304397595001360

[11] M. Reynolds, "A new rule for LTL tableaux," in *7th International Symposium on Games.* Open Publishing Association, 2016, pp. 287–301.

[12] M. D'Agostino, D. M. Gabbay, R. Hähnle, and J. Posegga, *Handbook of tableau methods.* Springer Science & Business Media, 2013.

[13] X. Chen, M.-T. Trinh, N. Rodrigues, L. Peña, and G. Roşu, "Towards a unified proof framework for automated fixpoint reasoning using matching logic," *Proceedings of the ACM on Programming Languages*, vol. 4, no. OOPSLA, pp. 1–29, 2020.

[14] D. Hilbert and W. Ackermann, "Grundzüge der theoretischen logik," *Bull. Amer. Math. Soc*, vol. 36, pp. 22–25, 1930.

[15] A. Church, "An unsolvable problem of elementary number theory," *American journal of mathematics*, vol. 58, no. 2, pp. 345–363, 1936.

[16] A. M. Turing, "On computable numbers, with an application to the Entscheidungsproblem," *Proceedings of the London mathematical society*, vol. 2, no. 1, pp. 230–265, 1937.

[17] F. P. Ramsey, "On a problem of formal logic," *Proceedings of the London Mathematical Society*, vol. 2, no. 1, pp. 264–286, 1930.

[18] M. Y. Vardi, "Why is modal logic so robustly decidable?" in *Descriptive Complexity and Finite Models: Proceedings of a DIMACS Workshop, January 14-17, 1996, Princeton University*, vol. 31. American Mathematical Soc., 1997, p. 149.

[19] E. A. Emerson and A. P. Sistla, "Deciding full branching time logic," *Information and Control*, vol. 61, no. 3, pp. 175–201, 1984.

[20] E. Gradel and I. Walukiewicz, "Guarded fixed point logic," in *Proceedings. 14th Symposium on Logic in Computer Science (Cat. No. PR00158).* IEEE, 1999, pp. 45–54.

[21] M. E. Stickel, "A unification algorithm for associative-commutative functions," *Journal of the ACM (JACM)*, vol. 28, no. 3, pp. 423–434, 1981.

[22] C. Hathhorn, C. Ellison, and G. Roşu, "Defining the undefinedness of C," in *Proceedings of the 36th ACM SIGPLAN Conference on Programming Language Design and Implementation*, 2015, pp. 336–345.

[23] B. Morrell, "Haskel syntax and static semantics written in K-framework," no. http://hdl.handle.net/2142/105410, 2018.

[24] E. Hildenbrandt, M. Saxena, N. Rodrigues, X. Zhu, P. Daian, D. Guth, B. Moore, D. Park, Y. Zhang, A. Stefanescu *et al.*, "KEVM: A complete formal semantics of the ethereum virtual machine," in *2018 IEEE 31st Computer Security Foundations Symposium (CSF).* IEEE, 2018, pp. 204–217.

[25] B. Courcelle and J. Engelfriet, *Graph structure and monadic second-order logic: a language-theoretic approach.* Cambridge University Press, 2012, vol. 138.

[26] I. Hodkinson, "Loosely guarded fragment of first-order logic has the finite model property," *Studia Logica*, vol. 70, no. 2, pp. 205–240, 2002.

[27] A. Tarski *et al.*, "A lattice-theoretical fixpoint theorem and its applications." *Pacific journal of Mathematics*, vol. 5, no. 2, pp. 285–309, 1955.

[28] G. Roşu, "Matching Logic," *Logical Methods in Computer Science*, vol. Volume 13, Issue 4, Dec. 2017. [Online]. Available: https://lmcs.episciences.org/4153

[29] W. W. Boone and H. Rogers, "On a problem of JHC Whitehead and a problem of Alonzo Church," *Mathematica Scandinavica*, vol. 19, no. 2, pp. 185–192, 1967.

[30] Y. N. Moschovakis, *Descriptive set theory.* American Mathematical Soc., 2009, no. 155.

[31] D. A. Martin, "Borel determinacy," *Annals of Mathematics*, pp. 363–371, 1975.

[32] Maude Team. The Maude system. [Online]. Available: maude.cs.illinois.edu/

[33] H. Ganzinger, G. Hagen, R. Nieuwenhuis, A. Oliveras, and C. Tinelli, "Dpll (t): Fast decision procedures," in *International Conference on Computer Aided Verification.* Springer, 2004, pp. 175–188.

.1 Decidability of $\mathrm{ML}^{\nexists\mu}$

Lemma 11. *Every* $\mathrm{ML}^{\nexists\mu}$*pattern is equivalent to some positive-form guarded pattern.*

Proof. Every $\mathrm{ML}^{\nexists\mu}$pattern may be transformed into an equivalant positive-form pattern by De Morgan's laws. Let φ be a positive-form pattern, we construct an equivalent guarded pattern.

Suppose $\varphi = \mu X.\alpha(X)$ and $\alpha(X)$ is a guarded pattern. Suppose X is unguarded in some sub-pattern $\alpha(X)$ of the form $\sigma Y.\beta(Y, X)$ and Y is guarded in $\sigma Y.\beta(Y, X)$. We may use the (KNASTER-TARSKI) proof rule of matching logic to obtain an equivalent (due to the (PRE-FIXEDPOINT) proof rule) pattern, $\beta(\sigma Y.\beta(Y, X), X)$. This pattern has all unguarded occurances outside the scope of the fixedpoint operator.

We may now transform this pattern to obtain a conjunctive normal form:

$$(X \vee \alpha_1(X)) \wedge \cdots \wedge (X \vee \alpha_n(X)) \wedge \beta'(X)$$

where all occurances of X in α_i and β' are guarded.

This is equivalant to:

$$(X \vee (\alpha_1(X) \wedge \cdots \wedge \alpha_n(X))) \wedge \beta'(X)$$

It is obvious that:

$$\begin{aligned}
&\mu X.(\alpha_1(X) \wedge \cdots \wedge \alpha_n(X)) \wedge \beta'(X) \\
\implies &\mu X.(X \vee (\alpha_1(X) \wedge \cdots \wedge \alpha_n(X))) \wedge \beta'(X)
\end{aligned}$$

Let $\gamma = (X \vee (\alpha_1(X) \wedge \cdots \wedge \alpha_n(X))) \wedge \beta'(X)$. It is enough to show that $\gamma(\mu X.\bar{\alpha}(X) \wedge \beta(X)) \implies \mu X.\bar{\alpha}(X) \wedge \beta'(X)$.

$$\begin{aligned}
&\gamma(\mu X.\bar{\alpha}(X) \wedge \beta(X)) \\
&\equiv((\mu X.\bar{\alpha}(X) \wedge \beta(X)) \vee \bar{\alpha}(\mu X.\bar{\alpha}(X) \wedge \beta(X))) \\
&\quad \wedge \beta(\mu X.\bar{\alpha}(X) \wedge \beta(X)) \\
\implies &((\bar{\alpha}(\mu X.\bar{\alpha}(X) \wedge \beta(X)) \wedge \beta(\mu X.\bar{\alpha}(X) \wedge \beta(X))) \\
&\quad \vee \bar{\alpha}(\mu X.\bar{\alpha}(X) \wedge \beta(X))) \\
&\quad \wedge \beta(\mu X.\bar{\alpha}(X) \wedge \beta(X)) \\
\implies &\bar{\alpha}(\mu X.\bar{\alpha}(X) \wedge \beta(X)) \wedge \beta'(\mu X.\bar{\alpha}(X) \wedge \beta(X)) \\
\implies &\mu X.\bar{\alpha}(X) \wedge \beta'(X)
\end{aligned}$$

□

Lemma 23. *Any infinite trace is either a μ-trace or a ν-trace.*

Proof. Any rule except (ONS) decreases the size of the trace label, so the (ONS) must apply infinitely often. Since there are only finite constants, there must be some oldest constant that is regenerated infinitely often. It may be either a μ-constant or a ν-constant. □

We show that for any positive guarded sentence ψ, it is satisfied in a model (i.e., its interpretation is nonempty) iff there exists a pre-model for ψ.

Definition 36. We extend the pattern syntax with two new constructs $\mu^\alpha X.\,\varphi$ and $\nu^\alpha X.\,\varphi$, where α is an ordinal. We define their semantics in M under the valuation ρ by transfinite induction as follows:

$$\begin{aligned}
\left|\mu^0 X.\,\varphi\right| &= \emptyset \\
\left|\mu^{\alpha+1} X.\,\varphi\right| &= |\varphi|_{\rho[|\mu^\alpha X.\,\varphi|/X]} \\
\left|\mu^\lambda X.\,\varphi\right| &= \bigcup_{\alpha<\lambda} |\mu^\alpha X.\,\varphi| \quad \text{for } \lambda \text{ limit ordinal} \\
\left|\nu^0 X.\,\varphi\right| &= M \\
\left|\nu^{\alpha+1} X.\,\varphi\right| &= |\varphi|_{\rho[|\nu^\alpha X.\,\varphi|/X]} \\
\left|\nu^\lambda X.\,\varphi\right| &= \bigcap_{\alpha<\lambda} |\nu^\alpha X.\,\varphi| \quad \text{for } \lambda \text{ limit ordinal}
\end{aligned}$$

Lemma 37. $|\mu X.\, \varphi| = \bigcup_\alpha |\mu^\alpha X.\, \varphi|$ *and* $|\nu X.\, \varphi| = \bigcap_\alpha |\nu^\alpha X.\, \varphi|$.

Let us extend the notion of definition lists given in Definition 12 by allowing equations of the form $U = \kappa^\alpha X.\, \varphi$ for $\kappa \in \{\mu, \nu\}$. Let us extend the expansion operator $\langle\!|\varphi|\!\rangle_{\mathcal{D}}$ accordingly.

Definition 38. Let ψ be a sentence, $\mathcal{D}$ be a definition list containing all definition constants in ψ, M be a model, and a be an element of M. Let $U_{k_1}, \ldots, U_{k_d}$ be the list of μ-constants in $\mathcal{D}$, ordered from the oldest to the youngest. If for some (irrelevant) valuation ρ we have $a \in |\langle\!|\psi|\!\rangle_{\mathcal{D}}|$, then we define the *signature ordinal sequence*, or simply the *signature* of ψ in a, written $\mathrm{Sig}_{\mathcal{D}}(\psi, a)$, as the least (in the lexicographical ordering) sequence of ordinals $(\alpha_1, \ldots, \alpha_d)$ such that $a \in |\langle\!|\psi|\!\rangle_{\mathcal{D}'}|$, where $\mathcal{D}'$ is obtained from $\mathcal{D}$ by replacing all equations of the form $U_{k_i} = \mu X.\, \varphi_{k_i}$ for $i \in \{1, \ldots, d\}$ with $U_{k_i} = \mu^{\alpha_i} X.\, \varphi_{k_i}$.

Lemma 39. $\mathrm{Sig}_{\mathcal{D}}(\psi, a)$ *as given in Definition 38 is well-defined.*

Note that $\mathrm{Sig}_{\mathcal{D}}(\psi, a)$ is defined when $a \in |\psi|$. For technical convenience, we define $\mathrm{Sig}_{\mathcal{D}}(\psi, a) = \infty$ when $a \notin |\psi|$ and assume ∞ is larger than all other ordinal sequences.

Lemma 40. *Let $\varphi_1, \varphi_2, \varphi$ be sentences whose definitions constants are in $\mathcal{D}$, M be a model, and a be an element of M. The following propositions hold.*

1. *If $a \in |\varphi_1 \wedge \varphi_2|$ then*

$$\mathrm{Sig}_{\mathcal{D}}(\varphi_1 \wedge \varphi_2, a) = \max\left(\mathrm{Sig}_{\mathcal{D}}(\varphi_1, a), \mathrm{Sig}_{\mathcal{D}}(\varphi_2, a)\right)$$

2. *If $a \in |\varphi_1 \vee \varphi_2|$ then*

$$\mathrm{Sig}_{\mathcal{D}}(\varphi_1 \vee \varphi_2, a) = \min\left(\mathrm{Sig}_{\mathcal{D}}(\varphi_1, a), \mathrm{Sig}_{\mathcal{D}}(\varphi_2, a)\right)$$

3. *If $a \in |\langle\varphi_1,\ \varphi_2\rangle|$ then*

$$\mathrm{Sig}_{\mathcal{D}}(\langle\varphi_1,\ \varphi_2\rangle, a) \geq \min_{(a_1,a_2)\in\bar{A}}\ \max_{i\in\{1,2\}} \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_i)$$

where $\bar{A} = \{(a_1, a_2) \mid a_1 \in |\langle\!|\varphi_1|\!\rangle|\,, a_2 \in |\langle\!|\varphi_2|\!\rangle|\,, a \in \mathrm{app}_M(a_1, a_2)\}$.

4. *If $a \in |[\varphi_1,\ \varphi_2]|$ then*

$$\mathrm{Sig}_{\mathcal{D}}([\varphi_1,\ \varphi_2], a) \geq \sup_{(a_1,a_2)\in\bar{A}}\ \min_{i\in\{1,2\}} \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_i)$$

where $\bar{A} = \{(a_1, a_2) \mid a_1 \in |\langle\!|\varphi_1|\!\rangle|\,, a_2 \in |\langle\!|\varphi_2|\!\rangle|\,, a \in \mathrm{app}_M(a_1, a_2)\}$.

5. *If $a \in |\mu X.\, \varphi|$ and $(U_i = \mu X.\, \varphi) \in \mathcal{D}$ is the ith μ-constant in $\mathcal{D}$, then $\mathrm{Sig}_{\mathcal{D}}(\mu X.\, \varphi, a)$ and $\mathrm{Sig}_{\mathcal{D}}(U_i, a)$ are the same at the first $(i-1)$ ordinals.*

6. *If $a \in |\nu X.\, \varphi|$ and $(V = \nu X.\, \varphi) \in \mathcal{D}$, then $\mathrm{Sig}_{\mathcal{D}}(\nu X.\, \varphi, a) = \mathrm{Sig}_{\mathcal{D}}(V, a)$;*

7. *If $a \in |U|$ and $(U_i = \mu X.\, \varphi_i) \in \mathcal{D}$ is the ith μ-constant in $\mathcal{D}$, then* $\mathrm{Sig}_{\mathcal{D}}(U, a) > \mathrm{Sig}_{\mathcal{D}}(\varphi[U/X], a)$*, and they are the same at the first $(i-1)$ ordinals.*

8. *If $a \in |V|$ and $(V = \nu X.\, \varphi) \in \mathcal{D}$, then*

$$\mathrm{Sig}_{\mathcal{D}}(V, a) = \mathrm{Sig}_{\mathcal{D}}(\varphi[V/X], a)$$

For any normal sequent $\Gamma = \{\varphi_1, \varphi_2\}$, we write $|\langle\!|\Gamma|\!\rangle_{\mathcal{D}}|$ to mean $\bigcap_i |\langle\!|\varphi_i|\!\rangle_{\mathcal{D}}|$ and drop $\mathcal{D}$ when it is understood from the context.

Definition 41. Given a pre-model (T, L) for ψ, we define a corresponding canonical model M as follows:

1. The carrier set M contains as elements all the leaves and $(\textsc{app}_1)$ nodes of T. For any $s \in \mathrm{Nodes}(T)$, we define by des_s its closest descendant node (may be itself) that belongs to M. Note that des_s is well-defined, because each infinite path in the pre-model must contain infinitely many $(\textsc{app}_1)$ nodes, since all patterns are guarded.

2. $a \in \mathrm{app}_M(a_1, a_2)$ for every non-constant symbol σ, iff a is an $(\textsc{app}_1)$ node, and $L(a)$ contains a pattern of the form $\langle \varphi_1,\ \varphi_2 \rangle$, and a has a child node s with $L(s) = L(a) \rightsquigarrow \langle \varphi_1,\ \varphi_m \rangle$, and s has exactly one child node s' with $L(s') = L(a) \rightsquigarrow \langle \varphi_1,\ \varphi_m \rangle \rightsquigarrow \mathrm{wit}$ for some $\mathrm{wit} \in \mathrm{Wit}(L(a), \sigma)$, and s' has exactly n child nodes denoted $s_1, \ldots, s_n$, and that $\mathrm{des}_{s_1} = a_1$, …, $\mathrm{des}_{s_n} = a_n$.

3. $c_M = \{s \in \mathrm{Nodes}(T) \mid c \in L(s)\}$.

Theorem 42. *$ML^{\nexists,\mu}$ is decidable.*

Lemma 37. *$|\mu X.\, \varphi| = \bigcup_\alpha |\mu^\alpha X.\, \varphi|$ and $|\nu X.\, \varphi| = \bigcap_\alpha |\nu^\alpha X.\, \varphi|$.*

Lemma 39 (,)**.** $\mathrm{Sig}_{\mathcal{D}}(\psi, a)$ *as given in Definition 38 is well-defined.*

Proof. Let us assume the notations given in Definition 38. Note that finite sequences of ordinals are well-founded. Therefore, we only need to show that there exists a sequence $(\alpha_1, \ldots, \alpha_n)$ such that $a \in |\langle\!|\psi|\!\rangle_{\mathcal{D}'}|$. The proof is standard and can be carried out by induction on n and the structural induction on ψ. □

Lemma 40. *Let $\varphi_1, \varphi_2, \varphi$ be sentences whose definitions constants are in $\mathcal{D}$, M be a model, and a be an element of M. The following propositions hold.*

1. *If $a \in |\varphi_1 \wedge \varphi_2|$ then*

$$\mathrm{Sig}_{\mathcal{D}}(\varphi_1 \wedge \varphi_2, a) = \max\left(\mathrm{Sig}_{\mathcal{D}}(\varphi_1, a), \mathrm{Sig}_{\mathcal{D}}(\varphi_2, a)\right)$$

2. *If $a \in |\varphi_1 \vee \varphi_2|$ then*

$$\mathrm{Sig}_{\mathcal{D}}(\varphi_1 \vee \varphi_2, a) = \min\left(\mathrm{Sig}_{\mathcal{D}}(\varphi_1, a), \mathrm{Sig}_{\mathcal{D}}(\varphi_2, a)\right)$$

3. *If* $a \in |\langle \varphi_1,\ \varphi_2 \rangle|$ *then*

$$\mathrm{Sig}_{\mathcal{D}}(\langle \varphi_1,\ \varphi_2 \rangle, a) \geq \min_{(a_1,a_2)\in \bar{A}} \max_{i\in\{1,2\}} \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_i)$$

where $\bar{A} = \{(a_1, a_2) \mid a_1 \in |\llparenthesis \varphi_1 \rrparenthesis|, a_2 \in |\llparenthesis \varphi_2 \rrparenthesis|, a \in \mathrm{app}_M(a_1, a_2)\}$*.*

4. *If* $a \in |[\varphi_1,\ \varphi_2]|$ *then*

$$\mathrm{Sig}_{\mathcal{D}}([\varphi_1,\ \varphi_2], a) \geq \sup_{(a_1,a_2)\in \bar{A}} \min_{i\in\{1,2\}} \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_i)$$

where $\bar{A} = \{(a_1, a_2) \mid a_1 \in |\llparenthesis \varphi_1 \rrparenthesis|, a_2 \in |\llparenthesis \varphi_2 \rrparenthesis|, a \in \mathrm{app}_M(a_1, a_2)\}$*.*

5. *If* $a \in |\mu X.\, \varphi|$ *and* $(U_i = \mu X.\, \varphi) \in \mathcal{D}$ *is the ith* μ*-constant in* $\mathcal{D}$*, then* $\mathrm{Sig}_{\mathcal{D}}(\mu X.\, \varphi, a)$ *and* $\mathrm{Sig}_{\mathcal{D}}(U_i, a)$ *are the same at the first* $(i-1)$ *ordinals.*

6. *If* $a \in |\nu X.\, \varphi|$ *and* $(V = \nu X.\, \varphi) \in \mathcal{D}$*, then* $\mathrm{Sig}_{\mathcal{D}}(\nu X.\, \varphi, a) = \mathrm{Sig}_{\mathcal{D}}(V, a)$*;*

7. *If* $a \in |U|$ *and* $(U_i = \mu X.\, \varphi_i) \in \mathcal{D}$ *is the ith* μ*-constant in* $\mathcal{D}$*, then* $\mathrm{Sig}_{\mathcal{D}}(U, a) > \mathrm{Sig}_{\mathcal{D}}(\varphi[U/X], a)$*, and they are the same at the first* $(i-1)$ *ordinals.*

8. *If* $a \in |V|$ *and* $(V = \nu X.\, \varphi) \in \mathcal{D}$*, then*

$$\mathrm{Sig}_{\mathcal{D}}(V, a) = \mathrm{Sig}_{\mathcal{D}}(\varphi[V/X], a)$$

Proof. We only prove (3) and (4). The other proofs are the same as in [10].

(3). Let $\bar{\alpha} = \mathrm{Sig}_{\mathcal{D}}(\langle \varphi_1,\ \varphi_2 \rangle)$ and $\mathcal{D}_{\bar{\alpha}}$ be $\mathcal{D}'$ as given in Definition 38. Then, we have $a \in |\llparenthesis \langle \varphi_1,\ \varphi_2 \rangle \rrparenthesis_{\mathcal{D}_{\bar{\alpha}}}|$. By the definition of expansion operator, we have $a \in |\sigma(\llparenthesis \varphi_1 \rrparenthesis_{\mathcal{D}_{\bar{\alpha}}}, \llparenthesis \varphi_2 \rrparenthesis_{\mathcal{D}_{\bar{\alpha}}})|$. Then, there exist a_1, a_2 such that $a \in \mathrm{app}_M(a_1, a_2)$ and $a_i \in \llparenthesis \varphi_i \rrparenthesis_{\mathcal{D}_{\bar{\alpha}}}$ for $i \in \{1, 2\}$. Let $\bar{\alpha}_i = \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_i)$. Then we have $\bar{\alpha}_i \leq \bar{\alpha}$. This implies that $\max_i \bar{\alpha}_i \leq \bar{\alpha}$. Therefore, we have

$$\mathrm{Sig}_{\mathcal{D}}(\langle \varphi_1,\ \varphi_2 \rangle) \geq \min_{(a_1,a_2)\in \bar{A}} \max_{i\in\{1,2\}} \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_i)$$

(4). Let $\bar{\alpha} = \mathrm{Sig}_{\mathcal{D}}([\varphi_1,\ \varphi_2])$ and $\mathcal{D}_{\bar{\alpha}}$ be $\mathcal{D}'$ as given in Definition 38. Then, we have $a \in |\llparenthesis [\varphi_1,\ \varphi_2] \rrparenthesis_{\mathcal{D}_{\bar{\alpha}}}|$. By the definition of expansion operator, we have $a \in |\bar{\sigma}(\llparenthesis \varphi_1 \rrparenthesis_{\mathcal{D}_{\bar{\alpha}}}, \ldots, \llparenthesis \varphi_n \rrparenthesis_{\mathcal{D}_{\bar{\alpha}}})|$. Then for all a_1, a_2 such that $a \in \mathrm{app}_M(a_1, a_2)$, there exists $i \in \{1, 2\}$ such that $a_i \in |\llparenthesis \varphi_i \rrparenthesis_{\mathcal{D}_{\bar{\alpha}}}|$, and thus $\mathcal{D}_{\bar{\alpha}} \geq \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_i)$. Therefore, $\mathcal{D}_{\bar{\alpha}} \geq \min_i \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_i)$ for every $a_1, \ldots, a_n$ such that $a \in \mathrm{app}_M(a_1, a_2)$, and we have

$$\mathrm{Sig}_{\mathcal{D}}([\varphi_1,\ \varphi_2]) \geq \sup_{(a_1,a_2)\in \bar{A}} \min_{i\in\{1,2\}} \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_i)$$

□

Proposition 27. *If a positive guarded sentence* ψ *is satisfied in* M *on* a*, there exists a pre-model for* ψ*.*

Proof. Let $\mathcal{D} = |\!\rangle \psi \langle\!|$ and $(U_{k_1} = \mu X.\, \psi_{k_1}); \ldots ; (U_{k_d} = \mu X.\, \psi_{k_d})$ be the sub-list of all μ-constants. We construct a quasi-model for ψ by selecting the rules from S_{mod} to construct a child node s. For every constructed node s, we associate an element $a_s \in M$, with the property that $a_s \in |\langle\!| L(s) |\!\rangle|$, if $L(s)$ is a normal sequent.

Firstly, we construct the root of T and for the associated element, we choose any element that matches ψ.

Suppose we have already constructed a node s and included it in the quasi-model, with the associated element a_s. We show how to proceed from this point, depending on what tableau rule may be applied:

1. $L(s)$ cannot be an inconsistent sequent, because $a_s \in |\langle\!| L(s) |\!\rangle|$.

2. If we can apply (AND), (ONS), (MU), or (NU) to s, we apply any of those rules. s has exactly one child s'. Let $a_{s'} = a_s$.

3. Otherwise if (OR-L) or (OR-R) can be applied to s, whose label is $L(s) = \varphi_1 \vee \varphi_2, \Gamma$, then s can have one of two child nodes s_1 and s_2, with labels $L(s_1) = \{\varphi_1\} \cup \Gamma$ and $L(s_2) = \{\varphi_2\} \cup \Gamma$, respectively.

   Let $i = \arg\min_i \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_s)$. We select the child node s_i and define $a_{s_i} = a_i$. We can prove that $a_i \in |\langle\!| \varphi_i |\!\rangle|$.

4. Otherwise if (APP_1) can be applied, then we construct all its child nodes.

5. Otherwise if (APP_2) can be applied to s, with label $L(s) = \Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle$, then we define

   $$(a_1, a_2) = \operatorname*{arg\,min}_{(a_1,a_2) \in \bar{A}} \max_{i \in \{1,2\}} \mathrm{Sig}_{\mathcal{D}}(\varphi_i, a_i)$$

   where $\bar{A} = \{(a_1, a_2) \mid a_1 \in |\langle\!| \varphi_1 |\!\rangle|, a_2 \in |\langle\!| \varphi_n |\!\rangle|, a_s \in \mathrm{app}_M(a_1, a_2)\}$. We define the witness function $\mathrm{wit}\colon \Gamma[\,] \to \{1, 2\}$ as follows. For every $[\psi_1,\ \psi_2] \in \Gamma[\,]$, since $a_s \in |\langle\!| [\psi_1,\ \psi_2] |\!\rangle|$ and $a_s \in \mathrm{app}_M(a_1, a_2)$, there exists $i \in \{1, 2\}$ such that $a_i \in |\langle\!| \psi_i |\!\rangle|$, and we define $\mathrm{wit}([\psi_1,\ \psi_2]) = i$. We construct s' whose label $L(s')$ is $\Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle \rightsquigarrow \mathrm{wit}$.

6. Otherwise (APP_3) applies to s with label is $L(s) = \Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle \rightsquigarrow \mathrm{wit}$ as defined in (5), we construct both child nodes of s, written s_1, s_2, and define $a_{s_j} = a_j$ for every $j \in \{1, 2\}$, where a_1, a_2 are defined in (5).

   We need to prove that $a_j \in |\langle\!| L(s_j) |\!\rangle|$. By definition, $L(s_j) = \{\varphi_j\} \cup \Gamma_j^{\mathrm{wit}}$. We have $a_j \in |\langle\!| \varphi_j |\!\rangle|$ by construction. For any $\psi_j \in \Gamma_j^{\mathrm{wit}}$, We have $a_j \in |\langle\!| \psi_j |\!\rangle|$ by the definition of Γ_j^{wit}.

Let us denote the resulting quasi-model as T_0. Next, we show that (T_0, L) is a pre-model for ψ.

Assume the opposite and suppose (T_0, L) is not a pre-model. Then there exists a μ-trace Tr on a path P of T. Suppose U_{k_i} is the oldest definition constant that regenerates infinitely often on Tr. Then there exists a node s on Tr such that $U_{k_1}, \ldots, U_{k_{i-1}}$ do not regenerate after s.

Consider the signature ordinals of the patterns on Tr after s. Using Lemma 40, we observe that the i-length prefixes of the signature ordinals never increase, and every regeneration of U_{k_i} strictly decrease the signatures at the ith position. Since ordinal sequences are well-founded, Tr cannot be infinite, which is a contradiction. Therefore, (T_0, L) contains no μ-trace, and thus it is a pre-model for ψ. □

Proposition 28. *If there exists a pre-model for a positive guarded sentence ψ then ψ is satisfiable.*

Proof. Suppose ψ has a pre-model (T, L) and M is its corresponding canonical model as defined in Definition 41. Let $s_0 = \mathrm{root}(T)$ be the root of T. Let $\mathcal{D} = \rrparenthesis\psi\llparenthesis$ and let $(V_{k_1} = \nu X.\, \psi_{k_1}); \ldots; (V_{k_d} = \nu X.\, \psi_{k_d})$ be the sub-list of ν-constants in $\mathcal{D}$. For the sake of contraction, we assume $\mathrm{des}_{s_0} \notin |\psi|$ for some (irrelevant) ρ.

Firstly, we define the notion of a ν-signature, $\mathrm{Sig}^{\nu}_{\mathcal{D}}(\varphi, a)$, which is defined for a sentence φ and $a \in M$ such that $a \notin |\llparenthesis\varphi\rrparenthesis|$, as follows:

$$\mathrm{Sig}^{\nu}_{\mathcal{D}}(\varphi, a) = \mathrm{Sig}^{\nu}_{\mathcal{D}}(\mathit{not}(\varphi), a)$$

where the definition list $\mathrm{not}(\mathcal{D})$ is obtained from $\mathcal{D}$ by replacing every definition $(U = \kappa X.\, \varphi)$ with $(U = \mathit{not}(\kappa X.\, \varphi))$. Recall that $\mathit{not}(\varphi)$ is the equivalent positive guarded formula of $\neg\varphi$, obtained by pushing in the negation. Note that Lemma 40 translates to ν-signatures after interchanging μ with ν, $\bar{\sigma}$ with σ, and $\vee$ with $\wedge$.

Next, we show that the assumption $\mathrm{des}_{s_0} \notin |\psi|$ implies that there exists a μ-trace on some path P of T, which contradicts the fact that (T, L) is a pre-model.

In the following, we construct P and a μ-trace Tr on P, simultaneously. The first pattern $\mathrm{Tr}(s_0) = \psi$. Now, suppose we have constructed Tr up to $\mathrm{Tr}(s)$ for some node s on P, such that $\mathrm{des}_s \notin |\llparenthesis\mathrm{Tr}(s)\rrparenthesis|$. We select the child node s' and $\mathrm{Tr}(s')$ as follows:

1. If s is an (AND)/(OR)/(MU)/(NU)/(ONS) node, then s has exactly one child node s' and
   (a) if $L(s)$ is not reduced, then $\mathrm{Tr}(s') = \mathrm{Tr}(s)$;
   (b) if $L(s) = \varphi_1 \wedge \varphi_2$ or $L(s) = \varphi_1 \vee \varphi_2$ is reduced, we let $i = \arg\min_i \mathrm{Sig}^{\nu}_{\mathcal{D}}(\varphi_i, \mathrm{des}_s)$ and define $\mathrm{Tr}(s') = \varphi_i$.
   (c) in other cases, let $\mathrm{Tr}(s')$ be the only resulting pattern.
2. If s is an (APP$_1$) node, and
   (a) if $L(s) = \{\langle\varphi_1,\ \varphi_2\rangle\} \cup \Gamma$ and $\mathrm{Tr}(s) = \langle\varphi_1,\ \varphi_2\rangle$, then by the hypothesis, $s \notin |\llparenthesis\langle\varphi_1,\ \varphi_2\rangle\rrparenthesis|$. Note that s has a child node s' with $L(s') = L(s) \rightsquigarrow \langle\varphi_1,\ \varphi_2\rangle$, which has exactly one child node s'' with

$L(s'') = L(s) \rightsquigarrow \langle\varphi_1,\ \varphi_2\rangle \rightsquigarrow \mathrm{wit}$ for some $\mathrm{wit} \in \mathrm{Wit}(L(s), \sigma)$, which has n child nodes denoted $s_1, \ldots, s_n$. By the construction of the canonical model, $s \in \mathrm{app}_M(\mathrm{des}_{s_1}, \mathrm{des}_{s_2})$. Therefore, there exists $i \in \{1, 2\}$ such that $\mathrm{des}_{s_i} \notin |\llparenthesis\varphi_i\rrparenthesis|$. Let $i = \arg\min_i \mathrm{Sig}^{\mu}_{\mathcal{D}}(\varphi_i, \mathrm{des}_{s_i})$ and we select $\mathrm{Tr}(s_i) = \varphi_i$.

(b) if $L(s) = \{[\varphi_1,\ \varphi_2]\} \cup \Gamma$ and $\mathrm{Tr}(s) = [\varphi_1,\ \varphi_2]$, then by the hypothesis, $s \notin |\llparenthesis[\varphi_1,\ \varphi_2]\rrparenthesis|$. Let

$$(\mathrm{des}_{s_1}, \mathrm{des}_{s_2}) = \mathop{\arg\min}_{(a_1,a_2)\in\bar{A}} \max_{i\in\{1,2\}} \mathrm{Sig}^{\nu}_{\mathcal{D}}(\varphi_i, \mathrm{des}_{s_i})$$

where $\bar{A} = \{(a_1, a_2) \mid s \in \mathrm{app}_M(a_1, a_2)\}$. We select any $i \in \{1, 2\}$ and let $\mathrm{Tr}(s_i) = \varphi_i$.

(c) For any other cases, stop the construction of Tr.

If the constructed trace Tr is finite, then either the last pattern $\mathrm{Tr}(s_N)$ is a constant symbol or its negation, or s_N is a leaf node and the pattern $\mathrm{Tr}(s_N)$ is a $\bar{\sigma}$-pattern. From the definition of the canonical model, we have $\mathrm{des}_{s_N} \in \mathrm{Tr}(s_N)$, a contradiction.

If Tr is infinite, then by a similar argument to the one in Proposition 27, we can show that the oldest definition constant that regenerates infinitely often on Tr is a μ-constant, which contradicts the fact that (T, L) is a pre-model.

Therefore, our assumption that $\mathrm{des}_{s_0} \notin |\psi|$ is incorrect, and thus ψ is satisfiable in the canonical model. □

Theorem 26. *For any positive guarded sentence ψ, there exists a pre-model for ψ iff ψ is satisfiable.*

Proof. By Propositions 27 and 28. □

Theorem 25. *For any positive guarded sentence ψ, determining whether ψ is satisfiable is decidable.*

Proof. By Theorem 26, we can look for a pre-model for ψ. We will construct a tree automaton Aut on infinite trees that accepts exactly the pre-models for ψ. Then by the Emerson-Jutla theorem, it is decidable to determine whether the language accepted by Aut is empty.

Aut is constructed in two steps. Firstly, we define an *outer automaton* Aut_o that accepts the quasi-models, which are essentially the *regular trees* generated by the set of tableau rules S_{mod}. Secondly, we define an *inner automaton* Aut_i that is a Büchi automaton on infinite words that accepts the μ-traces. Then, we combine the two automatons using the Safra deterministic construction and obtain a tree automaton that accepts precisely the pre-models for ψ. □

Theorem 42 (,). *$ML^{\nexists,\mu}$ is decidable.*

Proof. By Theorem 25. □

Theorem 31. *For an arbitary positive-form guarded pattern φ, there exists either a pre-model or a refutation.*

Proof. Define
$S_{\text{all}} = S_{\text{common}} \cup \{(\textsc{or}), (\textsc{app}_1), (\textsc{app'}_2), (\textsc{app}_3)\}$. Note that the ($\textsc{app'}_2$) and not ($\textsc{app}_2$) rule is used.

Construct a labeled tree T such that:

1. $L(\text{root}(T)) = \{\psi\}$;
2. For every $s \in \text{Nodes}(T)$, if $L(s)$ is an inconsistent sequent then s is a leaf of T;
3. For every $s \in \text{Nodes}(T)$, if $L(s)$ is not an inconsistent sequent and one of the tableau rules in S_{all} can be applied (with respect to the definition list $\mathcal{D} = \llbracket\psi\rrbracket$), and the resulting sequents are $\text{seq}_1, \ldots, \text{seq}_k$, then s has exactly k child nodes $s_1, \ldots, s_k$, and $L(s_1) = \text{seq}_1$, $\ldots$, $L(s_k) = \text{seq}_k$.

We define an infinite game for two players played on T. Player 1 will try to show that φ is satisfiable, and player 2 that it is not.

The game is played as follows:

- The game starts with the root node.
- In any ($\textsc{or}$) node, player 1 may choose any child.
- In any ($\textsc{app}_1$) node, player 2 may choose any child.
- In any ($\textsc{app'}_2$) node, player 1 may choose any child (i.e. any witness).
- In any ($\textsc{app}_3$) node, player 2 may choose any child.
- In all other non-leaf nodes, the only child is chosen.

The result of the game is a path of the tableau. If the path is finite it ends with sequent. If that sequent is consistent, then player 1 wins. Otherwise player 2 wins. If the path is infinite, player 2 wins iff there is a μ-trace.

A strategy exists for player 1 iff there is a pre-model contained in T. Intuitively, player 1 needs to pick all nodes that belong to the pre-model. The root of T obviously belongs to the pre-model. If the current node is in the pre-model and player 1 is to play, player 1 may pick the child node from the pre-model. If player 2 is to play, all child nodes are part of the pre-model, and so the choice does not matter.

By a similar argument we may show that a strategy exists for player 2 iff there is a refutation contained in T.

Clearly, only one player may have a winning strategy, and so a pattern may not have both a pre-model and a refutation. It is now left to show that either one must exist. We now show that the game always has a winning strategy for one player. This may be deduced from the theory of infinite games of Gale and Stewart [30].

A game $G(Y, A)$ is defined by an arena Y, and a winning set A, for the first player. Here, $Y \subseteq X*$ is a set of strings over a set S, closed under initial segments, and such that any string in Y has a prolongation in Y. Let $F(Y) \subseteq$

X^ω be infinite sequences with all finite prefixes in arena Y. The winning set A is a subset of $F(Y)$. Players 1 and 2 pick elements from X alternately, constructing an infinite seqeunce in $F(Y)$. At infinity, player 1 wins if the selected sequence is in A. Otherwise, player 2 wins.

The set $F(Y)$ may be equipped with a Cantor topology induced by the metric $d(u, v) = 2^{-n}$ where $u \neq v$ and n is the least position such that $u(n) \neq v(n)$. In [31], Martin showed that if A is a Borel set, then the game is determined – i.e. has a single winner.

It remains to choose X, Y and A so that our game may be presented as a Gale and Stewart game with a Borel winning condition. We choose X to be the set of all sequents that can appear in the tree for φ plus a dummy symbol. The arena Y can be obtained by extending the paths ending with a leaf with infinite occurances of the dummy symbol. The winning condition A is the set of paths with neither a μ-trace nor passing through an axiom sequent. Notice that the natural definition of this set involves an existential second-order quantifier. In order to show that A is Borel, we first observe that the set of all infinite sequents over X that do not contain μ traces and do not pass through axiom sequents is an ω-regular language and so is in Σ^0_2 in the Borel hierarchy over X^ω. Now A can be obtained by intersecting this set with $F(Y)$., and the last is a closed set since it is the set of all infinite paths of a tree. Thus A is a Borel subset of X^ω with the Cantor topology. In order to see that it is Borel aslo in $F(Y)$ observe that every $Z \subseteq F(Y)$ that is closed in X^ω is also closed in $F(Y)$. □

.2 LTL Tableau

Theorem 35. *Every pre-model corresponds to an LTL model.*

Proof. Let T be a pre-model. We need to construct an model (S, R, L) from T, where S is the set of states, R is the transition relation, L is the labeling function.

The AML theory of LTL only allows one symbol, $\circ$, in the left hand side of $\langle\, \rangle$, and $\neg\circ$ in the left hand side of $[\,]$. This means that there is only one possible witness that may produce a model — mapping all $[\,]$s to 2. So the composition of the (EQUIV) and the (APP$_1$), (APP$_2$) and (APP$_3$) rules results in two sets of descendant sequents. The first set of sequents have label $\circ$, whereas the other set has for their labels the set of patterns in the second argument of the $\langle\, \rangle$s and $[\,]$s of the original sequent. Pick any decendant sequent from the second set and discard the rest. Since none of the other tableau rules allow branching, we are left with a single path.

Let S_0 be set of (APP$_1$) in this path. Let $(n, n') \in R_0$ if there is a path from n to n' without using the (APP$_i$) rules. Let s_{i-1} be the ith node in the chain (i.e. s_0 is the first node in the chain). There are two cases:

1. The chain is infinite and there are no μ-traces. There must be some $i < j$ such that $L(s_i) = L(s_j)$, and s_i and s_j are (APP$_1$) nodes. Let $S = S_0$, $R = R_0$, and N be the lowest such i.

2. The chain is finite. Let N be the length of the chain. Let $S = S_0 \cup \{*\}$. Let $R = R_0 \cup \{(N-1, *), (*, *)\}$.

In either case, R defines a non-branching chain of nodes, that eventually loops back to the Nth node.

We want to define a family of sets Δ_i, that are the patterns we want to hold at each node. Let each Δ_i be the set of expansions of patterns in the labels of all nodes between applications of $(\textsc{app}_1)$. Let $L(S_i)$ be the set of atomic labels in Δ_i.

Lemma 43. *If $\langle \circ,\ \alpha \rangle \in \Delta_i$, then $\alpha \in \Delta_{i+1}$. If $[\neg\circ,\ \alpha] \in \Delta_i$, then $\alpha \in \Delta_{i+1}$.*

Proof. If $\langle \circ,\ \alpha \rangle \in \Delta_i$, no rule besides the $(\textsc{app}_3)$ removes it. If this $\langle \circ,\ \alpha \rangle$ is chosen by the $(\textsc{app}_1)$ rule, then the $(\textsc{app}_3)$ rule must produce a node with a label that includes α. Otherwise, the $(\textsc{equiv})$ generates $[\neg\circ,\ \alpha]$, and the second half of this proof applies.

If $[\neg\circ,\ \alpha] \in \Delta_i$, again, no rule besides the $(\textsc{app}_3)$ removes it. Note that because of $(\textsc{equiv})$, there must be at least one $\langle\ \rangle$ pattern, and $(\textsc{app}_3)$ *must* apply. Since $\mathrm{wit}([\neg\circ,\ \alpha]) = 2$ and $\circ$ is the only symbol in the signature, $(\textsc{app}_3)$ must produce a node with a label that includes α. □

Lemma 44. *Suppose $\alpha U \beta \equiv \mu X.\beta \vee (\alpha \wedge \langle \circ,\ X \rangle) \in \Delta_i$, then there is some $d \geq i$ such that $\beta \in \Delta_d$, and for all f, if $i \leq f < d$ then $\{\alpha, \alpha\, U\, \beta, \circ(\alpha\, U\, \beta)\} \subseteq \Delta_f$.*

Proof. Suppose $\alpha\, U\, \beta \equiv \mu X.\beta \vee (\alpha \wedge \langle \circ,\ X \rangle) \in \Delta_i$. Let U be the corresponding definitional constant. Then, the $(\textsc{ons})$ and $(\textsc{mu})$ rules must apply before applying either $(\textsc{or-l})$ or $(\textsc{or-r})$. Thus, either β or both $\langle \circ,\ U \rangle$ and α are in Δ_i. If $\langle \circ,\ U \rangle \in \Delta_i$, then by Lemma 43, $\alpha\, U\, \beta \in \Delta_i$. By induction, α, $\alpha\, U\, \beta$ and $\langle \circ,\ X \rangle$ must remain in Δ, until β is present.

It remains to show that β appears in some Δ_d. If the branch is finite, there must be some final node s_f where the $(\textsc{app}_1)$ has no decendants. This may only happen iff there are no $\langle \circ,\ \_ \rangle$ labeled sequents. In particular, $\langle \circ,\ X \rangle \notin \Delta_f$. Therefore, β must be in some Δ_d for $i \leq d \leq f$.

Consider the case when the branch is infinite. Suppose (for sake of contradiction), it $\nexists d.\beta \in \Delta_d$. Then, when simplifying $\alpha\, U\, \beta$, we must always choose the $(\textsc{or-l})$ rule. But this implies that a μ-trace exists! Contradication. □

Lemma 45. *If $\neg(\alpha\, U\, \beta) \equiv \nu X.\neg\alpha \wedge [\neg\circ,\ X] \in \Delta_i$, then either:*

1. *there is some $d \geq i$ such that $\neg\alpha, \neg\beta \in \Delta_d$ and for all f if $i \leq f < d$, then $\{\neg\beta, \neg(\alpha\, U\, \beta), \circ(\neg(\alpha\, U\, \beta))\} \subseteq \Delta_f$, or*
2. *for all f if $i \leq f$, then $\{\neg\beta, \neg(\alpha\, U\, \beta), \circ(\neg(\alpha\, U\, \beta))\} \subseteq \Delta_f$.*

Proof. Similar to 44. If the corresponding definitional constant regenerates finitely, then we get the same contradiction as if a μ-trace existed. Otherwise, the first part of the proof, shows that 2. must hold. □

Lemma 46. *For all $i \geq 0$, for all $\alpha \in \Delta_i$, if α is the translation of an LTL formula, then $(S, R, g), s_i \models \alpha$.*

Proof. We prove this by induction on the construction of α. since we are only considering positive-form patterns,

1. Case $\sigma, \neg\sigma$: by definition of L, $\alpha \in L(s_i)$ and $(S, R, g), s_i \models \alpha$.

2. Case $\alpha \wedge \beta$: This label is removed by $(\textsc{and})$, and is replaced with both α and β before the next $(\textsc{app}_1)$. By applying the inductive hypothesis, α and β must hold in s_i.

3. Case $\alpha \vee \beta$: This label is removed by $(\textsc{or-l})$ or $(\textsc{or-r})$, and is replaced with either α or β before the next $(\textsc{app}_1)$. By applying the inductive hypothesis, α or β must hold in s_i.

4. Case $\alpha\, U\, \beta$: By Lemma 44 and induction.

5. Case $\neg(\alpha\, U\, \beta)$: By Lemma 45 and induction.

6. Case $\Diamond\beta \equiv \mu X.\beta \vee \langle\circ,\ X\rangle$: This is reduced to either β (by $(\textsc{or-l})$) or $\circ U$ (by $(\textsc{or-r})$), where U is the corresponding definitional constant. If it is reduced to β, then the by the induction, β holds on the current state. Otherwise, $\Diamond\beta \in \Delta_{i+1}$ by Lemma 43, and by induction β holds in the next state.

7. Case $\Box\beta$: This is reduced to $\beta, \langle,\ \Box\beta\rangle$. By induction and Lemma 43, β holds for all future states.

□

This ends the proof. □

$$(\textsc{and})\ \frac{\varphi_1 \wedge \varphi_2, \Gamma}{\varphi_1, \varphi_2, \Gamma} \qquad (\textsc{mu})\ \frac{\mu X.\, \varphi, \Gamma}{U, \Gamma} \text{ where } (U = \mu X.\, \varphi) \in \mathcal{D}$$

$$(\textsc{ons})\ \frac{U, \Gamma}{\varphi[U/X], \Gamma} \begin{array}{ll} \text{where} & (U = \kappa X.\, \varphi) \in \mathcal{D} \\ \text{and} & \kappa \in \{\mu, \nu\} \end{array} \qquad (\textsc{nu})\ \frac{\nu X.\, \varphi, \Gamma}{U, \Gamma} \text{ where } (U = \nu X.\, \varphi) \in \mathcal{D}$$

(a) S_{common}

$$(\textsc{or-l})\ \frac{\varphi_1 \vee \varphi_2, \Gamma}{\varphi_1, \Gamma} \qquad \frac{\varphi_1 \vee \varphi_2, \Gamma}{\varphi_2, \Gamma}\ (\textsc{or-r})$$

$$(\textsc{app}_1)\ \frac{\Gamma}{\{\Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle \mid \langle \varphi_1,\ \varphi_2 \rangle \in \Gamma\}} \text{ where } \dagger \text{ holds}$$

$$(\textsc{app}_2)\ \frac{\Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle}{\Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle \rightsquigarrow \text{wit}} \text{ where wit} \in \text{Wit}(\Gamma, \sigma)$$

$$(\textsc{app}_3)\ \frac{\Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle \rightsquigarrow \text{wit}}{\varphi_1, \Gamma_1^{\text{wit}} \qquad \varphi_2, \Gamma_2^{\text{wit}}}$$

(b) $S_{\text{mod}} = S_{\text{common}} +$ the 4 rules above

$$(\textsc{or})\ \frac{\varphi_1 \vee \varphi_2, \Gamma}{\varphi_1, \Gamma \qquad \varphi_2, \Gamma}$$

$$(\textsc{app}'_1)\ \frac{\Gamma}{\Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle} \begin{array}{ll} \text{where} & \langle \varphi_1,\ \varphi_2 \rangle \in \Gamma, \\ \text{and} & \dagger \text{ holds} \end{array}$$

$$(\textsc{app}'_2)\ \frac{\Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle}{\{\Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle \rightsquigarrow \text{wit} \mid \text{wit} \in \text{Wit}(\Gamma, \sigma)\}}$$

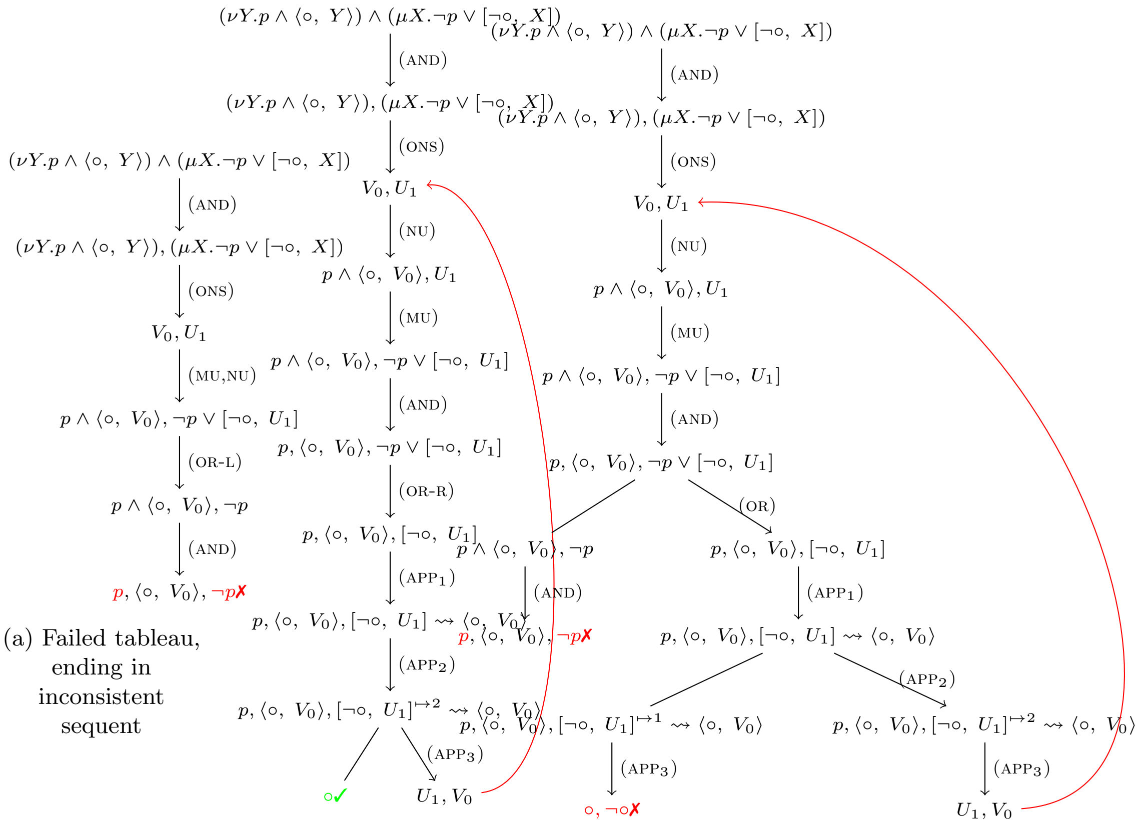
$$(\textsc{app}'_3)\ \frac{\Gamma \rightsquigarrow \langle \varphi_1,\ \varphi_2 \rangle \rightsquigarrow \text{wit}}{\varphi_i, \Gamma_i^{\text{wit}}} \text{ where } i \in \{1, 2\}$$

(c) $S_{\text{ref}} = S_{\text{common}} +$ the 4 rules above

Figure 8: S_{mod} and S_{ref} are two "dual" tableaux with respect to a definitional list $\mathcal{D}$. Γ is a finite nonempty set of sentences and $\varphi, \varphi_1, \varphi_2$ are sentences, whose definition constants are all contained in $\mathcal{D}$.

S_{mod} may be used for constructing a model for a pattern, and S_{ref} for constructing a refutation. They are dual in the sense that each set of non-deterministic rules in one (e.g. (OR-L) and (OR-R) in S_{mod}) are treated as a single rule with multiple conclusions in the other.

$\dagger$: In ($\textsc{app}_1$) and ($\textsc{app}'_1$), we require that Γ contains only constant symbols and their negations, applications, or dual applications. In other words, they can be applied only if all other rules cannot.

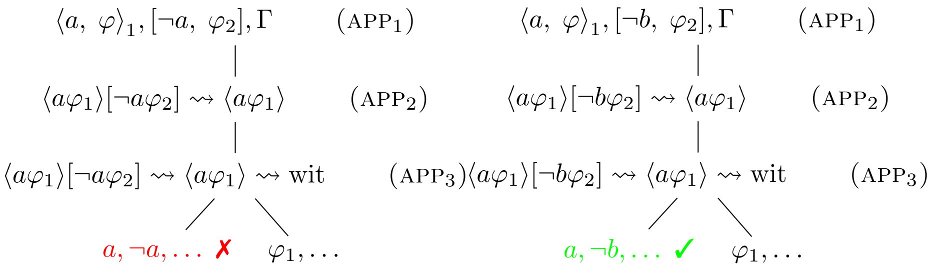

$(\nu Y.p \wedge \langle \circ,\ Y \rangle) \wedge (\mu X.\neg p \vee [\neg \circ,\ X])$
(AND)
$(\nu Y.p \wedge \langle \circ,\ Y \rangle), (\mu X.\neg p \vee [\neg \circ,\ X])$
(ONS)
V_0, U_1
(MU,NU)
$p \wedge \langle \circ,\ V_0 \rangle, \neg p \vee [\neg \circ,\ U_1]$
(OR-L)
$p \wedge \langle \circ,\ V_0 \rangle, \neg p$
(AND)
$p, \langle \circ,\ V_0 \rangle, \neg p$✗
$(\nu Y.p \wedge \langle \circ,\ Y \rangle) \wedge (\mu X.\neg p \vee [\neg \circ,\ X])$
(AND)
$(\nu Y.p \wedge \langle \circ,\ Y \rangle), (\mu X.\neg p \vee [\neg \circ,\ X])$
(ONS)
V_0, U_1
(NU)
$p \wedge \langle \circ,\ V_0 \rangle, U_1$
(MU)
$p \wedge \langle \circ,\ V_0 \rangle, \neg p \vee [\neg \circ,\ U_1]$
(AND)
$p, \langle \circ,\ V_0 \rangle, \neg p \vee [\neg \circ,\ U_1]$
(OR-R)
$p, \langle \circ,\ V_0 \rangle, [\neg \circ,\ U_1]$
(APP$_1$)
$p, \langle \circ,\ V_0 \rangle, [\neg \circ,\ U_1] \rightsquigarrow \langle \circ,\ V_0 \rangle$
(APP$_2$)
$p, \langle \circ,\ V_0 \rangle, [\neg \circ,\ U_1]^{\mapsto 2} \rightsquigarrow \langle \circ,\ V_0 \rangle$
$\circ$✓
(APP$_3$)
U_1, V_0
$(\nu Y.p \wedge \langle \circ,\ Y \rangle) \wedge (\mu X.\neg p \vee [\neg \circ,\ X])$
(AND)
$(\nu Y.p \wedge \langle \circ,\ Y \rangle), (\mu X.\neg p \vee [\neg \circ,\ X])$
(ONS)
V_0, U_1
(NU)
$p \wedge \langle \circ,\ V_0 \rangle, U_1$
(MU)
$p \wedge \langle \circ,\ V_0 \rangle, \neg p \vee [\neg \circ,\ U_1]$
(AND)
$p, \langle \circ,\ V_0 \rangle, \neg p \vee [\neg \circ,\ U_1]$
(OR)
$p \wedge \langle \circ,\ V_0 \rangle, \neg p$
(AND)
$p, \langle \circ,\ V_0 \rangle, \neg p$✗
$p, \langle \circ,\ V_0 \rangle, [\neg \circ,\ U_1]$
(APP$_1$)
$p, \langle \circ,\ V_0 \rangle, [\neg \circ,\ U_1] \rightsquigarrow \langle \circ,\ V_0 \rangle$
$p, \langle \circ,\ V_0 \rangle, [\neg \circ,\ U_1]^{\mapsto 1} \rightsquigarrow \langle \circ,\ V_0 \rangle$
(APP$_3$)
$\circ, \neg \circ$✗
(APP$_2$)
$p, \langle \circ,\ V_0 \rangle, [\neg \circ,\ U_1]^{\mapsto 2} \rightsquigarrow \langle \circ,\ V_0 \rangle$
(APP$_3$)
U_1, V_0

(a) Failed tableau, ending in inconsistent sequent

(b) Failed tableau, with μ-trace

(c) Refutation

Figure 29: Examples tableaux and refutations for $(\nu Y.p \wedge \langle \circ,\ Y \rangle) \wedge (\mu X.\neg p \vee [\neg \circ,\ X])$.
We have $\mathcal{D} = \{V_0 = \nu Y.p \wedge \langle \circ,\ Y \rangle, U_1 = \mu X.\neg p \vee [\neg \circ,\ X]\}$. The witness functions are denoted as superscripts over the corresponding patterns in the domain.


$\langle a,\ \varphi\rangle_1, [\neg a,\ \varphi_2], \Gamma$
(APP$_1$)
$\langle a\varphi_1\rangle[\neg a\varphi_2] \rightsquigarrow \langle a\varphi_1\rangle$
(APP$_2$)
$\langle a\varphi_1\rangle[\neg a\varphi_2] \rightsquigarrow \langle a\varphi_1\rangle \rightsquigarrow \text{wit}$
(APP$_3$)
$a, \neg a, \ldots$ ✗
$\varphi_1, \ldots$
$\langle a,\ \varphi\rangle_1, [\neg b,\ \varphi_2], \Gamma$
(APP$_1$)
$\langle a\varphi_1\rangle[\neg b\varphi_2] \rightsquigarrow \langle a\varphi_1\rangle$
(APP$_2$)
$\langle a\varphi_1\rangle[\neg b\varphi_2] \rightsquigarrow \langle a\varphi_1\rangle \rightsquigarrow \text{wit}$
(APP$_3$)
$a, \neg b, \ldots$ ✓
$\varphi_1, \ldots$

(a) Attempt to build a tableau for $\langle a,\ \varphi\rangle_1, [\neg a,\ \varphi_2], \Gamma$ by choosing $\text{wit}([\neg a,\ \varphi_2]) = 1$

(b) When building a tableau for $\langle a,\ \varphi\rangle_1, [\neg a,\ \varphi_2], \Gamma$, it is easiest to choose $\text{wit}([\neg a,\ \varphi_2]) = 1$.

Figure 32: Application of the (APP$_i$) rules to translations of modal μ-calculus formulae.

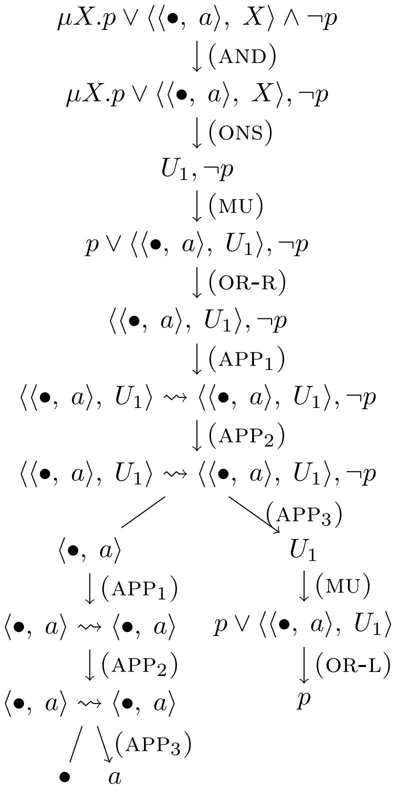
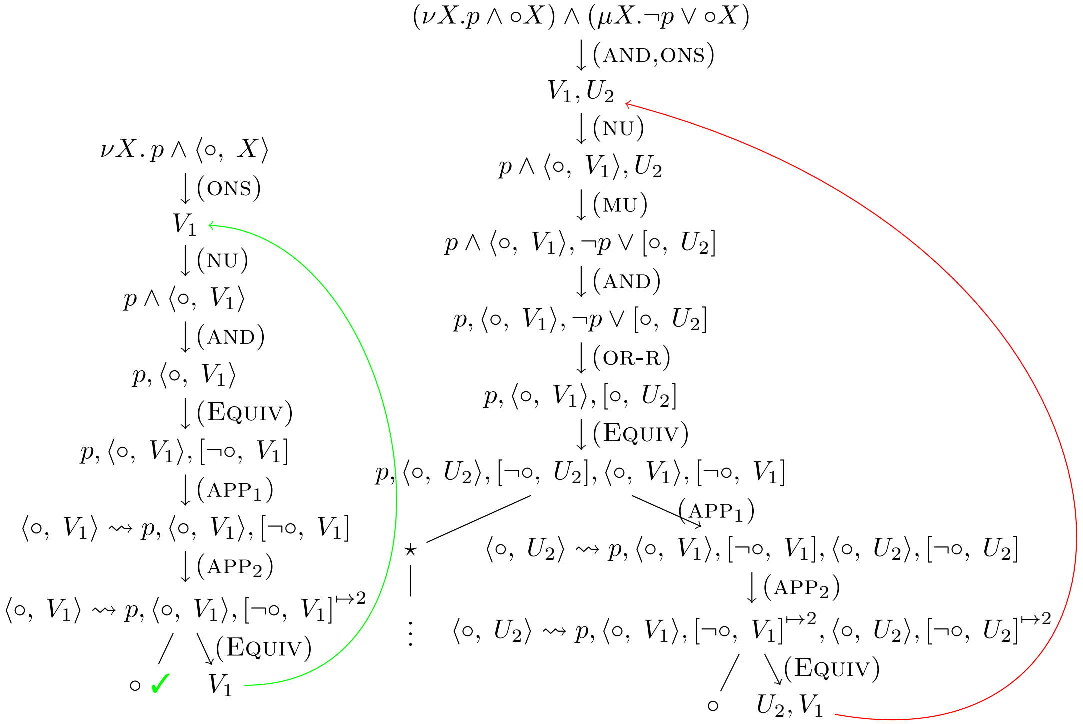

$\mu X.p \vee \langle\langle\bullet,\ a\rangle,\ X\rangle \wedge \neg p$
(AND)
$\mu X.p \vee \langle\langle\bullet,\ a\rangle,\ X\rangle, \neg p$
(ONS)
$U_1, \neg p$
(MU)
$p \vee \langle\langle\bullet,\ a\rangle,\ U_1\rangle, \neg p$
(OR-R)
$\langle\langle\bullet,\ a\rangle,\ U_1\rangle, \neg p$
(APP$_1$)
$\langle\langle\bullet,\ a\rangle,\ U_1\rangle \rightsquigarrow \langle\langle\bullet,\ a\rangle,\ U_1\rangle, \neg p$
(APP$_2$)
$\langle\langle\bullet,\ a\rangle,\ U_1\rangle \rightsquigarrow \langle\langle\bullet,\ a\rangle,\ U_1\rangle, \neg p$
(APP$_3$)
$\langle\bullet,\ a\rangle$
U_1
(APP$_1$)
(MU)
$\langle\bullet,\ a\rangle \rightsquigarrow \langle\bullet,\ a\rangle$
$p \vee \langle\langle\bullet,\ a\rangle,\ U_1\rangle$
(APP$_2$)
(OR-L)
$\langle\bullet,\ a\rangle \rightsquigarrow \langle\bullet,\ a\rangle$
p
(APP$_3$)
$\bullet$
a

Figure 33: Tableau for $\langle a*\rangle p \wedge \neg p \equiv \mu X.p \vee \langle\langle\bullet,\ a\rangle,\ X\rangle \wedge \neg p$.
$\mathcal{D} = \{U_1 = \mu X.p \vee \langle\langle\bullet,\ a\rangle,\ X\rangle\}$


$\nu X.\, p \wedge \langle \circ,\ X \rangle$
(ONS)
V_1
(NU)
$p \wedge \langle \circ,\ V_1 \rangle$
(AND)
$p, \langle \circ,\ V_1 \rangle$
(EQUIV)
$p, \langle \circ,\ V_1 \rangle, [\neg\circ,\ V_1]$
(APP$_1$)
$\langle \circ,\ V_1 \rangle \rightsquigarrow p, \langle \circ,\ V_1 \rangle, [\neg\circ,\ V_1]$
(APP$_2$)
$\langle \circ,\ V_1 \rangle \rightsquigarrow p, \langle \circ,\ V_1 \rangle, [\neg\circ,\ V_1]^{\mapsto 2}$
(EQUIV)
$\circ$ ✓
V_1
$(\nu X.p \wedge \circ X) \wedge (\mu X.\neg p \vee \circ X)$
(AND,ONS)
V_1, U_2
(NU)
$p \wedge \langle \circ,\ V_1 \rangle, U_2$
(MU)
$p \wedge \langle \circ,\ V_1 \rangle, \neg p \vee [\circ,\ U_2]$
(AND)
$p, \langle \circ,\ V_1 \rangle, \neg p \vee [\circ,\ U_2]$
(OR-R)
$p, \langle \circ,\ V_1 \rangle, [\circ,\ U_2]$
(EQUIV)
$p, \langle \circ,\ U_2 \rangle, [\neg\circ,\ U_2], \langle \circ,\ V_1 \rangle, [\neg\circ,\ V_1]$
(APP$_1$)
$\star$
$\vdots$
$\langle \circ,\ U_2 \rangle \rightsquigarrow p, \langle \circ,\ V_1 \rangle, [\neg\circ,\ V_1], \langle \circ,\ U_2 \rangle, [\neg\circ,\ U_2]$
(APP$_2$)
$\langle \circ,\ U_2 \rangle \rightsquigarrow p, \langle \circ,\ V_1 \rangle, [\neg\circ,\ V_1]^{\mapsto 2}, \langle \circ,\ U_2 \rangle, [\neg\circ,\ U_2]^{\mapsto 2}$
(EQUIV)
$\circ$
U_2, V_1

(a) Successful tableau for
$\Box p \equiv \nu X.p \wedge \circ X$.
$\mathcal{D} = \{V_1 = \nu X.p \wedge \circ X\}$

(b) Failed tableau for
$\Box p \wedge \Diamond \neg p \equiv (\nu X.p \wedge \circ X) \wedge (\mu X.\neg p \vee \circ X)$.
$\mathcal{D} = \{V_1 = \nu X.p \wedge \circ X, U_2 = \mu X.\neg p \vee \circ X\}$.
Note that the sequent at $\star$ is identical to that on the other (APP$_1$) branch.

Figure 34: Tableaux for translations of LTL formulae