MATCHING μ-LOGIC

XIAOHONG CHEN AND GRIGORE ROSU

Abstract. We propose in this paper matching μ-logic, an extension to matching logic with the fixpoint construct μ-binder. We investigate two notions of semantics of matching μ-logic: the standard semantics and the Henkin semantics. For the latter, a sound and complete proof system is given. We show that many important logics in mathematics and computer science, especially those about fixpoints and induction, are instances of matching μ-logic, including linear temporal logic (LTL), computation tree logic (CTL), modal μ-logic, propositional dynamic logic (PDL), and first-order logic with least fixpoints (FOL_{lfp}). We hope this paper offers some evidence that matching μ-logic can be taken as the unifying foundation of reasoning about fixpoints in mathematics and computer science.

Contents

(Xiaohong). Please, Xiaohong, please, revise the language. Definitions should be precise, rigorous, and compact. Proofs should be fluent, readable, intuitive, and, of course, sound.

1. Background and Motivation

2. Preliminaries

Definition 2.1 (Indexed families of sets)**.** Let S be a set. A set $F = \{F_i\}_{i\in I}$ is called an I-indexed family of sets over S where I is an index set and every F_i is a subset of S. When the concrete construction of the set S is not important, we omit it and simply assume it exists. The family F is called pairwise disjoint if for any $i, j \in I$ with $i \neq j$, $F_i \cap F_j = \emptyset$. When F is a pairwise disjoint family, we write $x \in F$ to mean that x belongs to some set F_i in the family.

Notation 2.2. *Let $F = \{F_i\}_{i\in I}$ and $G = \{G_i\}_{i\in I}$ be two families. An I-indexed function $h = \{h_i\}_{i\in I}$, denoted shortly as $h\colon F \to G$, is a set of functions $h_i\colon F_i \to G_i$ for every index $i \in I$. We use $\mathcal{P}(F) = \{\mathcal{P}(F_i)\}_{i\in I}$ to denote the family of the power sets of the element sets in F.*

Definition 2.3. Given a set M, a function $F\colon \mathcal{P}(M) \to \mathcal{P}(M)$ is order-preserving if for every sets $A, B \subseteq M$ such that $A \subseteq B$, $F(A) \subseteq F(B)$.

Theorem 2.4 (Knaster-Tarski theorem)**.** *Given a set M and an order-preserving function $F\colon \mathcal{P}(M) \to \mathcal{P}(M)$, the set of all fixpoints of F, denoted as* $\textsc{Fixpoint}(F) = \{A \subseteq M \mid F(A) = A\}$*, is nonempty and forms a complete lattice w.r.t. the subset relation $\subseteq$, with:*

- *$\mu F = \bigcap\{A \subseteq M \mid F(A) \subseteq A\}$ as the least fixpoint; and*
- *$\nu F = \bigcup\{A \subseteq M \mid A \subseteq F(A)\}$ as the greatest fixpoint.*

Theorem 2.5.

3. Syntax and Semantics of Matching μ-Logic

3.1. **Syntax.** We assume readers are familiar with first-order logic and modal μ-logic, in particular the use of $\exists$-binder and μ-binder. We also assume readers are familiar with the notions of many-sorted sets and functions.

Notation 3.1 (Variables)**.** *Let S be a nonempty set of sorts. We need two kinds of variables. One is* element variables*, denoted as $x{:}s, y{:}s, \ldots$. The set of all element variables is denoted as* EVar*. The other is* set variables*, denoted as $X{:}s, Y{:}s, \ldots$. The set of all set variables is denoted as* SVar*. To keep our notation tidy, we drop the sorts when they are clear from the context and just write x, y, X, Y, etc. We assume there are countably infinitely many element and set variables of each sort. Finally, let* $\textsc{Var} = \textsc{EVar} \cup \textsc{SVar}$ *be the set of all variables.*

Notation 3.2 (Many-sorted symbols)**.** *Let S be a set of sorts. We write $\sigma(s_1, \ldots, s_n):s$ to mean an n-ary symbol with argument sorts $s_1, \ldots, s_n$ and return sort s. We use Σ to denote the set of many-sorted symbols, and we sometimes write $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ to mean the same thing as $\sigma(s_1, \ldots, s_n):s$.*

Definition 3.3. Given a many-sorted signature $\mathbb{\Sigma} = (S, \textsc{Var}, \Sigma)$ with a sort set S, a variable set $\textsc{Var}$, and a many-sorted symbol set Σ, the set of *matching μ-logic patterns*, or simply *pattern*, denoted as $\textsc{Pattern} = \{\textsc{Pattern}_s\}_{s \in S}$, is inductively defined by the following grammar:

$$
\begin{array}{rcll}
\varphi_s & ::= & x{:}s \in \textsc{EVar}_s & \text{// Element variables} \\
& | & X{:}s \in \textsc{SVar}_s & \text{// Set variables} \\
& | & \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) \text{ where } \sigma(s_1, \ldots, s_n):s \in \Sigma & \text{// Structure} \\
& | & \neg\varphi_s & \text{// Complement} \\
& | & \varphi_s \wedge \varphi_s & \text{// Intersection} \\
& | & \exists x{:}s'.\varphi_s \text{ where } s' \in S & \text{// First-order binding} \\
& | & \mu X{:}s'.\varphi_s, \text{ where } s' \in S \text{ and } \varphi_s \text{ is positive in } X{:}s' & \text{// Least fixpoint binding}
\end{array}
$$

Notice that in $\exists x{:}s'.\varphi_s$ and $\mu X{:}s'.\varphi_s$, the sort s' can be different from s.

The syntax has two binders: $\exists$ and μ. The $\exists$-binder in $\exists x{:}s'.\varphi_s$ binds all free occurrences of $x{:}s'$ in φ_s, just like in first-order logic. The μ-binder in $\mu X{:}s'.\varphi_s$ binds all free occurrences of $X{:}s'$ in φ_s, just like in modal μ-logic.

When σ is a constant symbol, we write the pattern $\sigma()$ as just σ.

We use $\varphi[\varphi_s/x{:}s]$ to denote the result of substituting φ_s for $x{:}s$ in φ, where α-renaming happens implicitly to avoid unintended capturing of free variables. Similarly, we use $\varphi[\varphi_s/X{:}s]$ to denote the result of substituting φ_s for $X{:}s$ in φ, and α-renaming also happens implicitly.

Propositional connectives such as $\varphi \rightarrow \varphi$, $\varphi \vee \varphi$, $\varphi \leftrightarrow \varphi$ are defined in the usual way. The top $\top_s \equiv \exists x{:}s.x{:}s$ and bottom $\bot_s \equiv \mu X{:}s.X{:}s$ of sort s are defined in a way that they contain no free variables. It might not be clear why we define them as so, but we will see the reason very soon in Section 3.2 when we show that they have the intended semantics.

The $\exists$-binder and μ-binder have their dual versions defined in the following usual way:

$$\forall x{:}s.\varphi \equiv \neg\exists x{:}s.\neg\varphi \qquad\qquad \nu X{:}s.\varphi \equiv \neg\mu X{:}s.\neg\varphi[\neg X{:}s/X{:}s]$$

It is easy to show that the ν-binder is well-defined, by verifying that $\neg\varphi[\neg X{:}s/X{:}s]$ is positive in $X{:}s$ if φ is so.

Finally, we feel free to drop the sorts whenever we can.

3.2. **Standard semantics.**

Definition 3.4. A matching μ-logic model M of a give signature $\mathbb{\Sigma} = (S, \Sigma)$ consists of

- a sortwise family of carrier sets $M = \{M_s\}_{s \in S}$; and
- a function $\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \rightarrow \mathcal{P}(M_s)$ for every symbol $\sigma(s_1, \ldots, s_n):s \in \Sigma$.

We use the same notation M to mean the model and its carrier sets. Notice that symbols are interpreted as relations. We tacitly use the same notion σ_M for its pointwise extension $\sigma_M : \mathcal{P}(M_{s_1}) \times \cdots \times \mathcal{P}(M_{s_n}) \rightarrow \mathcal{P}(M_s)$, which is defined as

$$\sigma_M(A_1, \ldots, A_n) = \bigcup \{\sigma_M(a_1, \ldots, a_n) \mid a_1 \in A_1, \ldots, a_n \in A_n\} \tag{1}$$

$$\text{for all } A_1 \subseteq M_{s_1}, \ldots, A_n \subseteq M_{s_n}. \tag{2}$$

Remark 3.5. One can think of an n-ary symbol σ as being an $(n+1)$-ary predicate.

Remark 3.6. The pointwise extension of σ_M is convenient but also sets a trap. One might think that it can be any function $f\colon \mathcal{P}(M_{s_1})\times\cdots\times\mathcal{P}(M_{s_n}) \to \mathcal{P}(M_s)$. This is not true. For example, $\sigma_M(\emptyset,\ldots,\emptyset) = \emptyset$ always holds, no matter which model M we choose to interpret the symbol σ. In fact, σ_M can only be functions that are *extensional*, in the sense of (1).

In what follows, we give the first semantics notion for matching μ-logic: the *standard semantics*. The characteristic feature of the standard semantics is that it requires the least fixpoint pattern $\mu X.\varphi$ to be interpreted as the true least fixpoint in the model. Given that we have first-order binding $\exists$ in matching μ-logic, we know the standard semantics does not admit a complete proof system that can prove all valid patterns. We will formally prove this important result in Theorem **??**.

Definition 3.7. Given a model M, a valuation ρ is a function such that

- $\rho(x{:}s) \in M_s$ for all element variable $x{:}s \in \textsc{EVar}$;
- $\rho(X{:}s) \in \mathcal{P}(M_s)$ for all set variable $X{:}s \in \textsc{SVar}$.

Given an element variable x and an element $a \in M$, we define $\rho[a/x]$ as the valuation that maps x to a and maps all other variables the same as ρ. Similarly, $\rho[a_1/x_1,\ldots,a_n/x_n]$ is the valuation where we simultaneously substitute a_1 for x_1, …, a_n for x_n, where $x_1,\ldots,x_n$ all distinct. Given a set variable X and a subset $A \subseteq M$, we define $\rho[A/X]$ as the valuation that maps X to A and maps all other variables the same as ρ. We write $\rho \overset{x:s}{\sim} \rho'$ to mean that two valuations ρ and ρ' agree on all variables except $x{:}s$.

Definition 3.8. Given a model M and a valuation ρ, the *standard interpretation* is a function $\bar{\rho}_{\text{std}}\colon \textsc{Pattern} \to M$ defined inductively as follows:

- $\bar{\rho}_{\text{std}}(x{:}s) = \{\rho(x{:}s)\}$, for all $x{:}s \in \textsc{EVar}$;
- $\bar{\rho}_{\text{std}}(X{:}s) = \rho(X{:}s)$ for all $X{:}s \in \textsc{SVar}$;
- $\bar{\rho}_{\text{std}}(\sigma(\varphi_1,\ldots,\varphi_n)) = \sigma_M(\bar{\rho}_{\text{std}}(\varphi_1),\ldots,\bar{\rho}_{\text{std}}(\varphi_n))$, for all $\sigma(s_1,\ldots,s_n){:}s \in \Sigma$ and appropriate $\varphi_1,\ldots,\varphi_n$;
- $\bar{\rho}_{\text{std}}(\neg\varphi) = M_s \setminus \bar{\rho}_{\text{std}}(\varphi)$, for every $\varphi \in \textsc{Pattern}_s$;
- $\bar{\rho}_{\text{std}}(\varphi_1 \wedge \varphi_2) = \bar{\rho}_{\text{std}}(\varphi_1) \cap \bar{\rho}_{\text{std}}(\varphi_2)$, for every φ_1, φ_2 of the same sort;
- $\bar{\rho}_{\text{std}}(\exists x.\varphi) = \bigcup\{\overline{\rho'}_{\text{std}}(\varphi) \mid \text{for every } \rho' \overset{x}{\sim} \rho\}$.
- $\bar{\rho}_{\text{std}}(\mu X.\varphi) = \mu\mathcal{F}$, where $\mathcal{F}\colon \mathcal{P}(M) \to \mathcal{P}(M)$ is defined as $\mathcal{F}(A) = \overline{\rho[A/X]}_{\text{std}}(\varphi)$ for any subset $A \subseteq M$.

One can easily show the interpretation of $\mu X.\varphi$ is well-defined, by verifying that the function $\mathcal{F}$ is monotone, using the fact that φ is positive in X.

Example 3.9. Consider a matching μ-logic signature that contains one sort *Nat* for natural numbers and two symbols: 0:*Nat* and *s*(*Nat*):*Nat*. We can use μ-binder to specify an axiom that powers the induction reasoning over the domain as follows:

$$\text{(Inductive Domain)} \quad \mu X.(0 \vee s(X)).$$

Example 3.10. Consider a matching μ-logic signature that contains a sort *Nat* for natural numbers and a sort *Map* for finite maps, i.e., partial functions from *Nat* to *Nat* with finite domains. Apart from the usual axioms, we can use μ-binder to specify the following axiom that powers induction:

$$\text{(Inductive Domain)} \quad \mu M.(emp \vee (\exists x \exists y.\, x \mapsto y) * M)$$

So far, we have seen how we can use the μ-binder to define a *set* as a least fixpoint. In practice, however, this is not enough. For example, we want to define multiplication

$\_\times\_ : \Sigma_{NatNat,Nat}$ as

$$x \times y =_{\text{lfp}} ((x = 0) \wedge 0) \vee \exists x_1 . (x = s(x_1) \wedge y + (x_1 \times y))$$

For another example, we want to define a list abstraction on maps:

$$list(x) =_{\text{lfp}} ((x = 0) \wedge emp) \vee \exists y . (x \mapsto y * list(y))$$

We will see how to define such recursive symbols using the μ-binder in Section 5.

The interpretation given in Definition 3.8 is called the *standard semantics* because the μ-binder is interpreted as the least fixpoints in the model. One can prove, using Gödel's first incompleteness theorem, that matching μ-logic does not admit a complete deduction under the standard semantics. However, a complete deduction is possible under a different semantics notion, where the least fixpoint pattern $\mu X.\varphi$ is not asked to be interpreted as the true least fixpoint in the model, but only the least *definable* fixpoint. This is known as the *Henkin semantics*, which we defines in Section 3.3.

3.3. **Henkin semantics.**

(Xiaohong). Henkin semantics.

4. Proof System of Matching μ-Logic

(Proposition$_1$)	$\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Proposition$_2$)	$(\varphi_1 \rightarrow (\varphi_2 \rightarrow \varphi_3)) \rightarrow (\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \varphi_3)$
(Proposition$_3$)	$(\neg\varphi_1 \rightarrow \neg\varphi_2) \rightarrow (\varphi_2 \rightarrow \varphi_1)$
(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \rightarrow \varphi_2}{\varphi_2}$
(Variable Substitution)	$\forall x.\varphi \rightarrow \varphi[y/x]$
($\forall$)	$\forall x.(\varphi_1 \rightarrow \varphi_2) \rightarrow (\varphi_1 \rightarrow \forall x.\varphi_2)$ if $x \notin FV(\varphi_1)$
(Universal Generalization)	$\dfrac{\varphi}{\forall x.\varphi}$
(Propagation$_\bot$)	$C_\sigma[\bot] \rightarrow \bot$
(Propagation$_\vee$)	$C_\sigma[\varphi_1 \vee \varphi_2] \rightarrow C_\sigma[\varphi_1] \vee C_\sigma[\varphi_2]$
(Propagation$_\exists$)	$C_\sigma[\exists x.\varphi] \rightarrow \exists x.C_\sigma[\varphi]$ if $x \notin FV(C_\sigma[\exists x.\varphi])$
(Framing)	$\dfrac{\varphi_1 \rightarrow \varphi_2}{C_\sigma[\varphi_1] \rightarrow C_\sigma[\varphi_2]}$
(Pre-fixpoint)	$\varphi[\mu X.\varphi/X] \rightarrow \mu X.\varphi$
(Knaster-Tarski)	$\dfrac{\varphi[\psi/X] \rightarrow \psi}{\mu X.\varphi \rightarrow \psi}$
(Existence)	$\exists x.x$
(Singleton Variable)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$ where C_1 and C_2 are symbol contexts.

Figure 1. A sound and complete proof system of matching μ logic

Theorem 4.1. *The proof system in Figure 1 is sound and complete w.r.t. Henkin semantics.*

Theorem 4.2. *The proof system in Figure 1 is sound but not complete w.r.t. standard semantics.*

5. Recursive Symbols

The μ-binder in matching μ-logic may not seem to be powerful enough to some readers, especially for those who are familiar with first-order logic with least fixpoints (FOL_{lfp}). In FOL_{lfp}, one can write a formula

$$[\text{lfp}_{R,x,y}(x = y \vee \exists z.(E(x,z) \wedge R(z,y)))](u,v) \tag{3}$$

with free variables u and v, that defines the reflexive and transitive closure of the predicate E. In (3), the least fixpoint binder lfp binds all free occurrences of the predicate variable R and variables x and y in formula $x = y \vee \exists z.(E(x,z) \wedge R(z,y))$. The result, $[\text{lfp}_{R,x,y}(x = y \vee \exists z.(E(x,z) \wedge R(z,y)))]$, is a "predicate" (because it can be applied to u, v and returns a truth value) that satisfies the next equation about the predicate variable R:

$$R(x,y) =_{\text{lfp}} x = y \vee \exists z.(E(x,z) \wedge R(z,y)). \tag{4}$$

In addition, it is also the "smallest predicates" among all those that satisfy (4). This means that for any first-order logic formula $\varphi(x,y)$ with two distinguished free variables x, y, if

$$\varphi(x,y) =_{\text{lfp}} x = y \vee \exists z.(E(x,z) \wedge \varphi(z,y)). \tag{5}$$

holds, then

$$R(x,y) \rightarrow \varphi(x,y) \tag{6}$$

holds. We refer readers to [**?**] for a comprehensive discussion about FOL_{lfp} and other fixpoint logics.

It may seem to readers that the μ-binder in matching μ-logic is less powerful than the one in FOL_{lfp}. Indeed, the μ-binder in matching μ-logic can only bind *set variables*, as in $\mu X.\varphi$, but not *symbols variables*. Since that set variables can be regarded as variables that range over *constant symbols*, one can say the μ-binder in matching μ-logic allows us to define *recursive constant symbols*. There is obviously a gap between the μ-binder and defining general recursive symbols with any number of arguments. In this section, we aim at filling this gap, and show that we can just as well define general recursive symbols using the μ-binder. Before we dive into the technical details, let us first see some examples.

5.1. Some motivating examples from separation logic.

We start with some examples borrowed from separation logic with recursive predicates. It is known that (static) separation logic can be naturally regarded as theories in matching logic, and thus in matching μ-logic. We only give a brief definition here. A comprehensive discussion about matching logic and separation logic is given in [**?**]. To be clear: we are *not* going to discuss separation logic in what follows. The examples that we use throughout the section are about matching μ-logic.

Consider a many-sorted signature with sorts *Nat* and *Map*. Here *Nat* is the sort of natural numbers, and *Map* is the sort of *finite maps*:

$$\{f : \mathbb{N} \rightharpoonup \mathbb{N} \mid \text{dom}(f) \text{ is finite}\}.$$

The *empty map* is the partial function that is undefined everywhere, i.e., $\text{dom}(\bot_{Map}) = \emptyset$. We will call finite maps just maps throughout this section.

We have three constructor symbols in the many-sorted signature:

$$emp \in \Sigma_{\lambda,Map} \tag{7}$$

$$\_ \mapsto \_ \in \Sigma_{NatNat,Map} \tag{8}$$

$$\_ * \_ \in \Sigma_{MapMap,Map} \tag{9}$$

We use the constant symbol $emp{:}Map$ to denote the empty map. We use $x \mapsto y$ to denote the singleton map which maps x to y and is undefined everywhere else. We use $h_1 * h_2$ to denote the *separating conjunction*, or *merge*, of two maps h_1, h_2. When the domains of h_1 and h_2 are not disjoint, $h_1 * h_2$ equals the bottom pattern $\bot_{Map}$, meaning that no maps can match it, not even the empty map emp.

We are interested in defining maps that have particular structures. For example, we want to have a symbol $list \in \Sigma_{Nat,Map}$ that takes a natural number x and returns the set of all maps that consist of a single linked-list starting at x. Intuitively speaking, we want $list$ to be the solution of the following equation about a unary symbol σ:

$$\sigma(x) = (emp \wedge x = 0) \vee \exists y . x \mapsto y * \sigma(y). \tag{10}$$

Not just that, we want it to be the "least solution" among all those that satisfy (10). Let us denote this fact as

$$list(x) =_{\mathrm{lfp}} (x = 0 \wedge emp) \vee \exists y . (x \mapsto y * list(y)). \quad \text{// just a notation} \tag{11}$$

Unfortunately, we cannot *define* the symbol $list$ as in $\mathrm{FOL}_{\mathrm{lfp}}$ as

$$list(x) = [\mathrm{lfp}_{\sigma,x}(emp \wedge x = 0) \vee \exists y . x \mapsto y * \sigma(y)](x), \quad \text{// not a valid definition} \tag{12}$$

because the right-hand side is beyond the syntax of matching μ-logic.

As another example, we want to have a symbol $lsegleft \in \Sigma_{NatNat,Map}$ that takes two natural numbers x (as the starting position) and y (as the ending position), and returns the set of all maps that consist of a singe linked-list from x to y. We can write it as

$$lsegleft(x, y) =_{\mathrm{lfp}} (x = y \wedge emp) \vee \exists z . (x \mapsto z * lsegleft(z, y)). \quad \text{// just a notation} \tag{13}$$

The name $lsegleft$ is a shortcut for "list segments recursively defined on the left-hand side". A dual version is $lsegright$, meaning "list segments recursively defined on the right-hand side". We can write it as

$$lsegright(x, y) =_{\mathrm{lfp}} (x = y \wedge emp) \vee \exists z . (lsegright(x, z) * z \mapsto y). \quad \text{// just a notation} \tag{14}$$

Intuitively, the two ways to define list segments are equivalent. In other words, we expect that

$$\forall x \forall y . lsegleft(x, y) = lsegright(x, y) \tag{15}$$

holds.

So far, (11), (13), and (14) remain an intuitive notation instead of a rigorous definition. In what follows, we will formally define these intuitive notations using the μ-binder in matching μ-logic, and we will prove the property (15).

5.2. **An important isomorphism.** We introduce a way to "mimic" an n-ary symbol with a constant symbol. This will let us define general recursive symbols as we saw in Section 5.1. The idea is in fact not new. In simply-typed λ-calculus with product types, a function f that takes two natural numbers and returns a boolean value is just a constant $f : (Nat \times Nat \to Bool)$. We adopt the same approach in matching μ-logic.

Proposition 5.1. *Given two sets A and B, there is an isomorphism:*

$$[A \to \mathcal{P}(B)] \underset{uncurry}{\overset{curry}{\rightleftarrows}} \mathcal{P}(A \times B) \tag{16}$$

defined as

$$curry(f) = \{(a, b) \mid b \in f(a)\}, \quad \text{for any } f : A \to \mathcal{P}(B) \tag{17}$$

$$uncurry(\alpha)(a) = \{b \mid (a, b) \in \alpha\}, \quad \text{for any } \alpha \subseteq A \times B. \tag{18}$$

Definition 5.2. Let $s, t \in S$ be two sorts, not necessarily distinct. Define a new sort $s \times t$ called the *product sort* of s and t. Two product sorts $s \times t$ and and $s' \times t'$ are the same if s is the same sort as s', and t is the same sort as t'. Define a new symbol $\langle \_, \_ \rangle_{s,t} \in \Sigma_{st, s\times t}$ called *pairing*, parametric on s and t. Define a new symbol $\_[\_]_{s,t} \in \Sigma_{s\times t s, t}$ called *projection*, parametric on s and t. We often drop the sort subscripts to keep our notation tidy. We add the following axiom:

$$\langle x, \varphi \rangle [y] = (x = y) \wedge \varphi. \tag{19}$$

Notation 5.3. *Let $s_1, \ldots, s_n, t \in S$ be $n+1$ sorts for some $n \geq 2$, not necessarily distinct. We write $s_1 \times \cdots \times s_n \times t$ to mean the product sort $s_1 \times (s_2 \times (\cdots \times (s_n \times t) \ldots))$. Let $\varphi_{s_1 \times \cdots \times s_n \times t}, \varphi_{s_1}, \ldots, \varphi_{s_n}, \varphi_t$ be patterns of the subscripted sorts. We write $\langle \varphi_1, \ldots, \varphi_n, \varphi_t \rangle$ to mean the pattern $\langle \varphi_1, \langle \ldots, \langle \varphi_n, \varphi_t \rangle \ldots \rangle \rangle$. We write $\varphi_{s_1 \times \cdots \times s_n \times t}[\varphi_1, \ldots, \varphi_n]$ to mean the pattern $\varphi_{s_1 \times \cdots \times s_n \times t}[\varphi_1] \ldots [\varphi_n]$.*

Proposition 5.4. *The notations $\langle \varphi_{s_1}, \ldots, \varphi_{s_n}, \varphi_t \rangle$ and $\varphi_{s_1 \times \cdots \times s_n \times t}[\varphi_{s_1}, \ldots, \varphi_{s_n}]$ defined in Notation 5.3 "behave like symbols", meaning that they satisfy the three propagation rules and the frame rule:*

- $\Gamma \vdash \langle \varphi_1, \ldots, \varphi_n, \varphi_t \rangle \leftrightarrow \bot$ *if any of* $\varphi_1, \ldots, \varphi_n, \varphi_t$ *is* $\bot$.
- $\Gamma \vdash \langle \varphi_1, \ldots, \varphi_i \vee \varphi_i', \ldots, \varphi_n, \varphi_t \rangle \leftrightarrow \langle \varphi_1, \ldots, \varphi_i, \ldots, \varphi_n, \varphi_t \rangle \vee \langle \varphi_1, \ldots, \varphi_i', \ldots, \varphi_n, \varphi_t \rangle$
- $\Gamma \vdash \langle \varphi_1, \ldots \varphi_n, \varphi_t \vee \varphi_t' \rangle \leftrightarrow \langle \varphi_1, \ldots \varphi_n, \varphi_t \rangle \vee \langle \varphi_1, \ldots \varphi_n, \varphi_t' \rangle$
- *TBC.*

5.3. **Separation logic examples revisited.** With product sorts, we can give a formal definition of the notations we used in define (11), (13), and (14) as follows:

$$\mathit{list}(x) \equiv \underbrace{(\mu\alpha . \exists x . \langle x, (x = 0 \wedge \mathit{emp}) \vee (\exists y . x \mapsto y * \alpha[y]) \rangle)}_{\alpha_{\mathit{list}}}[x] \tag{20}$$

$$\mathit{lsegleft}(x, y) \equiv \underbrace{(\mu\alpha . \exists x \exists y . \langle x, (x = y \wedge \mathit{emp}) \vee (\exists z . x \mapsto z * \alpha[z, y]) \rangle)}_{\alpha_{\mathit{lsegleft}}}[x, y] \tag{21}$$

$$\mathit{lsegright}(x, y) \equiv \underbrace{(\mu\alpha . \exists x \exists y . \langle x, (x = y \wedge \mathit{emp}) \vee (\exists z . \alpha[x, z] * z \mapsto y) \rangle)}_{\alpha_{\mathit{lsegright}}}[x, y] \tag{22}$$

Notice that *list*, *lsegleft*, *lsegright* are not actually symbols, but merely a notation. However, it is a good notation, in the sense that we can indeed regard $\mathit{list}(x), \mathit{lsegleft}(x, y), \mathit{lsegright}(x, y)$ as patterns that are symbols applied to arguments. When it comes to semantics, they are indeed interpreted as "the least symbol" that satisfies (11), (13), and (14).

Take *list* as an example. We can prove that it "satisfies" all three propagation rules.

$$\mathit{list}(\bot) = \alpha_{\mathit{list}}[\bot] = \bot, \tag{23}$$

$$\mathit{list}(\varphi_1 \vee \varphi_2) = \alpha_{\mathit{list}}[\varphi_1 \vee \varphi_2] = \alpha_{\mathit{list}}[\varphi_1] \vee \alpha_{\mathit{list}}[\varphi_2], \tag{24}$$

$$\mathit{list}(\exists x . \varphi) = \alpha_{\mathit{list}}[\exists x . \varphi] = \exists x . \alpha_{\mathit{list}}[\varphi]. \tag{25}$$

In addition, the (Framing) rule also holds:

$$\text{if } \varphi_1 \to \varphi_2 \text{ then } \alpha_{\mathit{list}}[\varphi_1] \to \alpha_{\mathit{list}}[\varphi_2], \text{ which is exactly } \mathit{list}(\varphi_1) \to \mathit{list}(\varphi_2). \tag{26}$$

The (Singleton Variables) rule also holds. We leave it as an exercise to interested readers. Last but not the least, we can prove the following two rules hold, which justifies the least

$$
\begin{array}{ll}
 & list(x) \to \psi \\
\text{by Notation (20)} \Longleftarrow & \alpha_{list}[x] \to \psi \\
\text{by (Plugout)} \Longleftarrow & \alpha_{list} \to \underbrace{\exists \alpha . (\alpha \wedge \lfloor \alpha[x] \to \psi \rfloor)}_{\alpha_0} \\
\text{by FOL reasoning} \Longleftarrow & \alpha_{list} \to \forall x . \alpha_0 \\
\text{by Notation (20)} \Longleftarrow & \mu\alpha . \exists x . \langle x, (x = 0 \wedge emp) \vee (\exists y . x \mapsto y * \alpha[y]) \rangle \to \forall x . \alpha_0 \\
\text{by (Knaster-Tarski), } y \notin FV(\psi) \Longleftarrow & \exists x . \langle x, (x = 0 \wedge emp) \vee (\exists y . x \mapsto y * (\forall x . \alpha_0)[y]) \rangle \to \forall x . \alpha_0 \\
\text{by FOL reasoning} \Longleftarrow & \langle x, (x = 0 \wedge emp) \vee (\exists y . x \mapsto y * (\forall x . \alpha_0)[y]) \rangle \to \forall x . \alpha_0 \\
\text{by FOL reasoning} \Longleftarrow & \langle x, (x = 0 \wedge emp) \vee (\exists y . x \mapsto y * \alpha_0[y/x][y]) \rangle \to \forall x . \alpha_0 \\
\text{by (Collaspe)} \Longleftarrow & \langle x, (x = 0 \wedge emp) \vee (\exists y . x \mapsto y * \psi[y/x]) \rangle \to \forall x . \alpha_0 \\
\text{by FOL reasoning, for some fresh } z \Longleftarrow & \langle x, (x = 0 \wedge emp) \vee (\exists y . x \mapsto y * \psi[y/x]) \rangle \to \alpha_0[z/x] \\
\text{by (Plugin)} \Longleftarrow & \langle x, (x = 0 \wedge emp) \vee (\exists y . x \mapsto y * \psi[y/x]) \rangle [z] \to \psi[z/x] \\
\text{by Axiom (19)} \Longleftarrow & (x = z) \wedge ((x = 0 \wedge emp) \vee (\exists y . x \mapsto y * \psi[y/x]) \to \psi[z/x] \\
\text{by FOL reasoning} \Longleftarrow & (x = 0 \wedge emp) \vee (\exists y . x \mapsto y * \psi[y/x]) \to \psi \\
\text{by assumption} \Longleftarrow & \text{QED}
\end{array}
$$

FIGURE 2. The (KNASTER-TARSKI) "proof rule", shown in (28), is sound for *list*.

fixpoint essence of *list*:

$$(x = 0 \wedge emp * \exists y . x \mapsto y * list(y)) \to list(x) \tag{27}$$

$$\frac{(x = 0 \wedge emp * \exists y . x \mapsto y * \psi[y/x]) \to \psi}{list(x) \to \psi} \text{ if } y \notin FV(\psi) \tag{28}$$

We sketch the proof of (28) in Figure 2.

5.4. **Define general recursive symbols.** We generalize the results and insights from the separation logic examples in Section 5.3 and propose a notation to define and use general recursive symbols in matching μ-logic.

Definition 5.5. Let $\mathbb{\Sigma} = (S, \Sigma)$ be a signature, and Γ be an axiom set of $\mathbb{\Sigma}$-patterns. Let $s_1, \ldots, s_n, t \in S$ be $n+1$ sorts, not necessarily distinct. Let $\sigma(s_1, \ldots, s_n){:}t$ be a symbol that is not in the signature $\mathbb{\Sigma}$, and denote the extended signature $\mathbb{\Sigma}'$. Let $x_1 : s_1, \ldots, x_n : s_n$ be n distinct variables, and ψ be a $\mathbb{\Sigma}'$-pattern that is positive in σ. Variables $x_1, \ldots, x_n$ may occur free in ψ. The pattern ψ may contain other free variables, too. We use the notation

$$\sigma(x_1, \ldots, x_n) =_{\text{lfp}} \psi \tag{29}$$

to mean the following:

(1) We define a new product sort $s_1 \times \cdots \times s_n \times t$ and the corresponding pairing and projection symbols as in Section 5.2. Notice that when $n \geq 2$, there are multiple

sorts being defined, because $s_1 \times \cdots \times s_n \times t$ is merely a notation to the product sort $s_1 \times (s_2 \times (\cdots \times (s_n \times t) \ldots))$. In other words, we are defining n new product sorts in total. They are $s_n \times t$, $s_{n-1} \times s_n \times t$, $\ldots$, $s_1 \times \cdots \times s_n \times t$.

(2) We add the new sorts and symbols defined in (1) to the signature $\mathbb{\Sigma}$, and denoted the extended signature $\mathbb{\Sigma}^\sigma$. Notice that $\mathbb{\Sigma}^\sigma$ is not $\mathbb{\Sigma}'$. The signature $\mathbb{\Sigma}^\sigma$ does not contain the symbol σ.

(3) We define an alias $\alpha_\sigma \equiv \mu\alpha.(\exists x_1 \ldots \exists x_n.\langle x_1, \ldots, x_n, \psi[\alpha/\sigma]\rangle)$, where α is a fresh set variable of sort $s_1 \times \cdots \times s_n \times t$, and $\psi[\alpha/\sigma]$ is a $\mathbb{\Sigma}^\sigma$-pattern obtained by repeatedly replacing every pattern of the form $\sigma(\varphi_1, \ldots, \varphi_n)$ in ψ with $\alpha_\sigma[\varphi_1, \ldots, \varphi_n]$, until there is no occurrence of the symbol σ anymore. It is easy to verify that $\psi[\alpha/\sigma]$ is a well-defined and well-formed $\mathbb{\Sigma}^\sigma$-pattern, and it is positive in α. Therefore, α_σ is a well-defined and well-formed $\mathbb{\Sigma}^\sigma$-pattern of sort $s_1 \times \ldots s_n \times t$.

(4) We define an alias $\sigma(\varphi_1, \ldots, \varphi_n) \equiv \alpha_\sigma[\varphi_1, \ldots, \varphi_n]$ for every $\mathbb{\Sigma}^\sigma$-patterns $\varphi_1, \ldots, \varphi_n$ of appropriate sorts.

The following two propositions show that the alias $\sigma(\varphi_1, \ldots, \varphi_n) \equiv \alpha_\sigma[\varphi_1, \ldots, \varphi_n]$ given in Definition 5.5 is a good one, in the sense that it indeed behaves as a recursive symbol.

Proposition 5.6. *Let $\sigma(x_1, \ldots, x_n) =_{\text{lfp}} \psi$. Then σ satisfies all three propagation rules,* (Framing) *rule,* (Singleton Variables) *rule,* (Pre-Fixpoint) *rule, and* (Knaster-Tarski) *rule.*

6. Instance: Modal Logic and Its Variants

In this section, we give a list of important logics and calculi about program reasoning that can be defined as matching logic theories or notations. The main point is to convince that matching logic is capable of serving as a unified logic for program verification, that allows us to reason about any properties written in any logics, about any programs written in any programming languages.

6.1. Hybrid Modal Logic.

As shown in Section 17.1.2, the modal logic S5 is definable in matching logic. In this section, we show that a first-order extension of modal logic, called hybrid modal logic, is also definable in matching logic. In particular, we show that the famous (Barcan) axioms are special cases of rule (Propagation$_\exists$) for unary symbols.

The syntax of hybrid modal logic contains a set SVar of state variables, a set Nom of nominals, and a binder $\forall$ that binds a state variable in a formula:

$$\varphi ::= p \in \text{AP} \mid x \in \text{SVar} \mid i \in \text{Nom} \mid \neg\varphi \mid \varphi \wedge \varphi \mid \Box\varphi \mid \forall x.\varphi \text{ where } x \in \text{SVar}$$

As in first-order logic, we define $\exists x.\varphi \equiv \neg(\forall x.\neg\varphi)$. As in modal logic S5, we define $\Diamond\varphi \equiv \neg(\Box\neg\varphi)$.

A hybrid modal logic model $\mathcal{M} = (S, R, V)$ is a triple where S is a nonempty set of states, R is a binary relation on S, and $V\colon \text{AP} \cup \text{Nom} \to \mathcal{P}(S)$ is a valuation. The valuation V is called a standard valuation if $V(i)$ is a singleton set for every $i \in \text{Nom}$. A model is standard if its valuation is standard. An assignment $g\colon \text{SVar} \to \mathcal{P}(S)$ is called standard if $g(x)$ is a singleton set for every $x \in \text{SVar}$. The semantics is defined inductively as follows:

- $\mathcal{M}, g, s \vDash_{hybrid} p$ if $s \in V(p)$ where $p \in \text{AP}$;
- $\mathcal{M}, g, s \vDash_{hybrid} x$ if $x \in g(x)$ where $x \in \text{SVar}$;
- $\mathcal{M}, g, s \vDash_{hybrid} i$ if $i \in V(i)$ where $i \in \text{Nom}$;
- $\mathcal{M}, g, s \vDash_{hybrid} \neg\varphi$ if $\mathcal{M}, g, s \nvDash_{hybrid} \varphi$;
- $\mathcal{M}, g, s \vDash_{hybrid} \varphi_1 \wedge \varphi_2$ if $\mathcal{M}, g, s \vDash_{hybrid} \varphi_1$ and $\mathcal{M}, g, s \vDash_{hybrid} \varphi_2$;

- $\mathcal{M}, g, s \vDash_{hybrid} \Box\varphi$ if $\mathcal{M}, g, s' \vDash_{hybrid} \varphi$ for every s' such that sRs';
- $\mathcal{M}, g, s \vDash_{hybrid} \forall x.\varphi$ if $\mathcal{M}, g', s \vDash_{hybrid} \varphi$ for every g' such that $g' \overset{x}{\sim} g$.

A hybrid modal logic formula is valid if $\mathcal{M}, g, s \vDash_{hybrid} \varphi$ holds for any standard model $\mathcal{M}$, standard assignment g, and state s A sound and complete proof system of hybrid modal logic is proposed in [], as shown in the following. We write $\vdash_{hybrid} \varphi$ if φ is provable in hybrid modal logic.

Proof system of hybrid modal logic extends propositional calculus with the following:

(K)	$\Box(\varphi \to \psi) \to (\Box\varphi \to \Box\psi)$	(N)	$\dfrac{\varphi}{\Box\varphi}$
(Q_1)	$\forall x.(\varphi \to \psi) \to (\varphi \to \forall x.\psi)$, if $x \notin FV(\varphi)$	(Gen)	$\dfrac{\varphi}{\forall x.\varphi}$
(Q_2-Svar)	$\forall x.\varphi \to \varphi[y/x]$	(Q_2-Nom)	$\forall x.\varphi \to \varphi[i/x]$
(Barcan)	$\forall x.\Box\varphi \to \Box\forall x.\varphi$		
(Nom)	$\forall x.(\Diamond^m(x \wedge \varphi) \to \Box^n(x \to \varphi))$, for every $m, n \in \mathbb{N}$	(Name)	$\exists x.x$

We can define a matching logic theory $\mathsf{hybridML} = (\mathbb{\Sigma}, H)$ that faithfully captures hybrid modal logic. The matching logic signature $\mathbb{\Sigma} = (\text{SVar} \cup \text{Nom}, \{state\}, \Sigma)$ where the variable set contains all state variables and nominals and the sort set contains exactly one sort *state* of states. The symbol set Σ consists of

- a unary symbol $\Diamond \in \Sigma_{state,state}$; Define the derived construct $\Box\varphi \equiv \neg(\Diamond\neg\varphi)$;
- a constant symbol $p \in \Sigma_{\lambda,state}$ for every atomic proposition $p \in \text{AP}$;

The axiom set $H = \emptyset$. With the above definitions,

Any hybrid modal logic formula φ is a matching logic pattern in theory $\mathsf{hybridML}$.

In addition, the state/valuation models of hybrid modal logic are essentially identical to the matching logic models of theory $\mathsf{hybridML}$, as summarized as follows.

Hybrid modal logic	Matching logic theory $\mathsf{hybridML}$
state set S	carrier set S
binary relation $R \subseteq S \times S$	interpretation of $\Diamond$ such that $s \in \Diamond_{\mathcal{M}}(t)$ if and only if sRt
standard valuation $V\colon \text{AP} \to \mathcal{P}(S)$	interpretation of p such that $p_{\mathcal{M}} = V(p)$
standard valuation $V\colon \text{Nom} \to \mathcal{P}(S)$	valuation $\rho\colon \text{Nom} \to S$ such that $V(i) = \{\rho(i)\}$
standard valuation $g\colon \text{SVar} \to \mathcal{P}(S)$	valuation $\rho\colon \text{SVar} \to S$ such that $g(x) = \{\rho(x)\}$
$\mathcal{M}, g, s \vDash_{hybrid} \varphi$	$s \in \bar{\rho}(\varphi)$

Immediately, we have the following conservative extension result for hybrid modal logic.

Theorem 6.1 (Conservative Extension for Hybrid Modal Logic)**.** *Let φ be a hybrid modal logic formula. Then $\vdash_{hybrid} \varphi$ if and only if $\vDash_{hybrid} \varphi$ if and only if* $\mathsf{hybridML} \vDash \varphi$ *if and only if* $\mathsf{hybridML} \vdash \varphi$.

We point out that the famous (N) and (Barcan) axioms in hybrid modal logic are in fact special cases of (Propagation$_\bot$) and (Propagation$_\exists$) rules in matching logic where the symbol takes exactly one argument. For any symbol $\sigma \in \Sigma_{s,s}$, define its complement $\bar{\sigma}(\varphi) \equiv \neg\sigma(\neg\varphi)$. We can show the following propositions hold for any pattern set H in matching logic:

- $H \vdash \varphi$ implies $H \vdash \bar{\sigma}(\varphi)$;
- $\emptyset \vdash \forall x\bar{\sigma}(\varphi) \to \bar{\sigma}(\forall x.\varphi)$.

6.2. **Polyadic modal logic.** ...

7. Instance: Modal μ-Logic

Mu-calculus is the extension of modal logic with induction and fixpoints. The syntax of mu-calculus is parametric on a set Var of variables, a set AP of atomic propositions, and a set Label of labels. Mu-calculus formulas are defined inductively as follows.

$$\varphi ::= p \in \text{AP} \mid x \in \text{Var} \mid \varphi \wedge \varphi \mid \neg\varphi \mid [a]\varphi \mid \mu x.\varphi \text{ if } x \text{ does not occur negatively in } \varphi$$

where $a \in$ Label. As in matching logic, define $\nu x.\varphi \equiv \neg\mu x.(\neg(\varphi[\neg x/x]))$. Notice that if x does not occur negatively in φ, so does $\neg(\varphi[\neg x/x])$. Define $\langle a\rangle\varphi \equiv \neg[a](\neg\varphi)$.

Mu-calculus formulas are interpreted on structures. A structure $\mathcal{M} = (S, R, V)$ consists of a set S of states, a family set $R = \{R_a\}_{a\in\text{Label}}$ with a binary relation $R_a \subseteq S \times S$ for each label a, and a valuation $V\colon \text{AP} \to \mathcal{P}(S)$. Given a structure $\mathcal{M}$ and an assignment $g\colon \text{Var} \to \mathcal{P}(S)$, the interpretation function $[\![\_]\!]_{\mathcal{M},g}$ is defined inductively as follows.

- $[\![p]\!]_{\mathcal{M},g} = V(p)$;
- $[\![x]\!]_{\mathcal{M},g} = g(x)$;
- $[\![\varphi_1 \wedge \varphi_2]\!]_{\mathcal{M},g} = [\![\varphi_1]\!]_{\mathcal{M},g} \cap [\![\varphi_2]\!]_{\mathcal{M},g}$;
- $[\![\neg\varphi]\!]_{\mathcal{M},g} = S \setminus [\![\varphi]\!]_{\mathcal{M},g}$;
- $[\![[a]\varphi]\!]_{\mathcal{M},g} = \{s \mid \text{for all } t \text{ such that } sR_at,\ t \in [\![\varphi]\!]_{\mathcal{M},g}\}$;
- $[\![\mu x.\varphi]\!]_{\mathcal{M},g} = \bigcap\{X \subseteq S \mid [\![\varphi]\!]_{\mathcal{M},g'} \subseteq X \text{ where } g' \overset{x}{\sim} g \text{ and } g(x) = X\}$.

A mu-calculus formula φ is valid, written as $\vDash_\mu \varphi$, if $[\![\varphi]\!]_{\mathcal{M},g} = S$ for every structure $\mathcal{M} = (S, R, V)$ and assignment g. Sound and complete deduction for mu-calculus remained open for more than a decade, until Walukiewicz showed in [**?**] that the following proof system, initially given by Kozen [**?**], is indeed complete. We write $\vdash_\mu \varphi$ if φ is provable in mu-calculus.

Proof system of mu-calculus extends propositional calculus with the following:

$$\text{(K)}\quad [a](\varphi \to \psi) \to ([a]\varphi \to [a]\psi) \qquad (\text{Mu}_1)\quad \varphi[(\mu x.\varphi)/x] \to \mu x.\varphi \qquad (\text{Mu}_2)\quad \frac{\varphi[\psi/x] \to \psi}{\mu x.\varphi \to \psi}$$

The way mu-calculus is defined in matching logic is similar as other modal logics. The theory $\mathsf{Mu} = (\Sigma, H)$ is a single-sorted theory and contains all definitions needed for the fixpoint constructs μ and ν. The variable set is Var. The signature set Σ contains

- a unary symbol a for every label $a \in$ Label;
- a constant symbol p for every atomic proposition $p \in$ AP;
- a constant symbol x for every variable $x \in$ Var.

Notice that we have both the variable x and the symbol x in matching logic. This is because in mu-calculus, a variable is either a standalone formula or the binding variable of a fixpoint construct. Mu-calculus assignments assign variables to sets, while matching logic valuations map variables to values (singleton sets). Therefore, we use the matching logic symbol x if it is a standalone formula in mu-calculus, and use the variable x if it is a binding variable in $\mu x.\varphi$ or $\nu x.\varphi$. In addition, we define $\langle a\rangle\varphi \equiv a(\varphi)$ and $[a]\varphi \equiv \neg a(\neg\varphi)$. With the above definitions and notations,

Any mu-calculus formula φ is a closed matching logic pattern in theory Mu.

There is a one-to-one correspondence between mu-calculus structures and the intended models of theory Mu, as summarized in the following. Since all mu-calculus formulas are closed, it does not matter what matching logic valuation we use.

Mu-calculus	Matching logic theory Mu with intended semantics
state set S	carrier set S
binary relation $R_a \subseteq S \times S$	interpretation of a such that $s \in a_{\mathcal{M}}(t)$ if and only if sR_at
valuation $V\colon \mathrm{AP} \to \mathcal{P}(S)$	interpretation of p such that $p_{\mathcal{M}} = V(p)$
assignment $g\colon \textsc{Var} \to \mathcal{P}(S)$	interpretation of x such that $x_{\mathcal{M}} = g(x)$
least fixpoint $[\![\mu x.\varphi]\!]_{\mathcal{M},g}$	intended interpretation $\bar{\rho}(\mu x.\varphi) = \mu[\![\varphi]\!]_{\mathcal{M},\rho}$ where ρ is any valuation (it makes no difference which ρ we use)
$s \in [\![\varphi]\!]_{\mathcal{M},g}$	$s \in \bar{\rho}(\varphi)$

Like hybrid modal logic (see Section 6.1), mu-calculus and the theory Mu admit the conservative extension result. Unlike hybrid modal logic, the proof involves both modal-theoretic and proof-theoretic approaches. The reason is that the above one-to-one correspondence only works for the *intended models*, not *all models*, of theory Mu. Therefore, we can conclude only that $\mathsf{Mu} \vDash \varphi$ implies $\vDash_\mu \varphi$, but not the other direction, as there may exist an unintended model, say $\mathcal{M}'$,which satisfies Mu and fails φ. Such unintended models $\mathcal{M}'$ are not considered at all in mu-calculus.

A proof-theoretic approach fills the gap. Careful readers may already notice that all axioms and rules in mu-calculus are provable in matching logic. Axiom (K) is provable as shown in Section 6.1. Axiom ($\textsc{Mu}_1$) is proved using ($\textsc{Fix}_\mu$), and rule ($\textsc{Mu}_2$) is proved using (Lfp). Therefore, we conclude that $\vdash_\mu \varphi$ implies $\mathsf{Mu} \vdash \varphi$,because we can mimic every mu-calculus proofs in the matching logic theory Mu. The conservative extension for mu-calculus is then obtained by completeness of both mu-calculus and matching logic.

We point out that the above reasoning, as illustrated in Figure 3, is very general. In the following subsections, we will use the same technique to prove the conservative extension results for LTL, CTL, CTL*, etc, so it is important to understand what is needed for the proof. As shown in Figure 3, the diagram involves 7 steps, with 4 of them are either proved by definition, or established results about matching logic. Among the rest 3 steps, Step ($\Longrightarrow_2$) requires to show all axioms and proof rules of mu-calculus are provable in theory Mu, which is not trivial and can involve some intelligence. Step ($\Longrightarrow_6$) requires to show all mu-calculus models can be regarded as matching logic models (often the intended models) of theory Mu; the proof is often by carrying out structural induction on formulas, and thus we often need to prove both directions. Notice that there is a game we can play when defining theory Mu. We can add more axioms to Mu, which makes Step ($\Longrightarrow_2$) easier to prove but Step ($\Longrightarrow_6$) harder, as it rules out more models. If we do not have enough axioms, Step ($\Longrightarrow_2$) may not be provable.Finally, we rely on the completeness of mu-calculus (Step ($\Longrightarrow_1$)), which is also an established result but far from trivial.

Theorem 7.1 (Conservative extension for mu-calculus)**.** *Let φ be a mu-calculus formula, and* Mu *be the matching logic theory for mu-calculus defined as above. Then, $\vdash_\mu \varphi$ if and only if* $\mathsf{Mu} \vdash \varphi$.

8. Instance: Transition Systems

Transition systems are important in computer science. They are used to model various types of hardware and software systems, and many algorithms and techniques are proposed to analyze transition systems and to reason about their properties. In mathematics, a transition system $\mathcal{M} = (M, A, \{\xrightarrow{a}\}_{a\in A})$ is a carrier set M and a label set A, accompanied with a binary relation $\xrightarrow{a} \subseteq M \times M$ for every label a. When A contains exactly one label, we write $\mathcal{M} = (M, \to)$ and call the transition system *unlabeled*.

This subsection is dedicated to defining transition systems in matching logic.

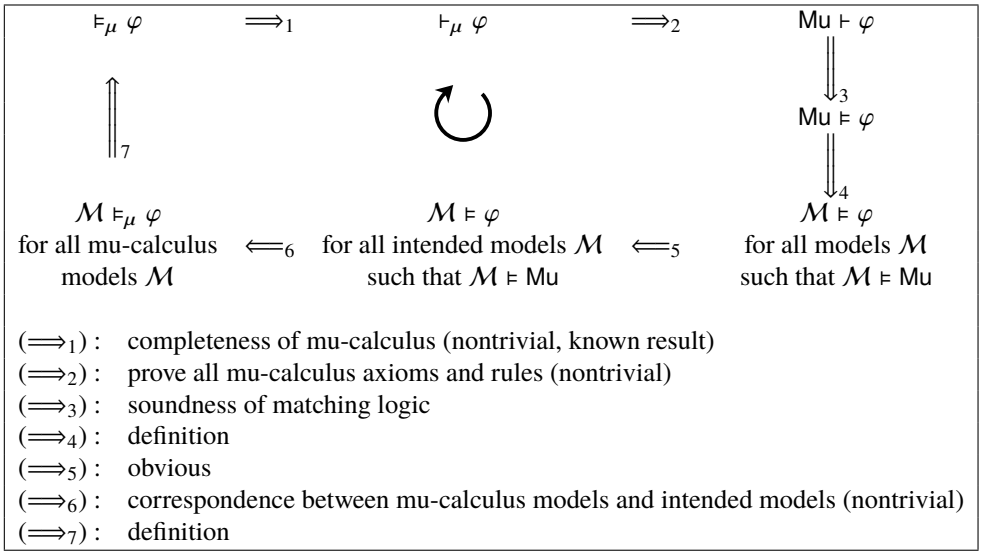

FIGURE 3. A general method to prove conservative extension, using mu-calculus as an example.

8.1. **Strong next and weak next.** Let us first consider unlabeled transition systems. Given an unlabeled transition system $\mathcal{M} = (M, \rightarrow)$. There are two ways to capture the transition relation $\rightarrow$. One is to define a predecessor function $pred\colon M \rightarrow \mathcal{P}(M)$ such that $pred(t) = \{s \mid s \rightarrow t\}$. The other is to define a successor function $succ\colon M \rightarrow \mathcal{P}(M)$ such that $succ(s) = \{t \mid s \rightarrow t\}$. It is easy to show that $succ(s) = \{t \mid s \in pred(t)\}$. Define a unary matching logic symbol $\bullet$ and interpret it to the predecessor function $\bullet_{\mathcal{M}}(t) = pred(t)$. We write $\bullet\varphi$ instead of $\bullet(\varphi)$. Define a derived construct $\overline{\bullet}(\varphi) \equiv \exists x.(x \wedge \lfloor \varphi \rightarrow \bullet(\varphi) \rfloor)$, and one can prove that $\overline{\bullet}$ is interpreted to the successor function $\overline{\bullet}_{\mathcal{M}}(s) = succ(s)$. We call the symbol $\bullet$ "strong next" and the construct $\overline{\bullet}$ "strong previous", and we will see why. Since "strong next" and "strong previous" are similar, let us only discuss the "strong next" $\bullet$.

Assume ρ is any valuation and $\bar{\rho}\colon$ PATTERN $\rightarrow \mathcal{P}(M)$ is the corresponding extended valuation that interprets patterns to sets. Let $s \in M$ and φ be a pattern. The symbol $\bullet$ is called "strong next", because $s \in \bar{\rho}(\bullet\varphi)$ if and only if *there exists* $t \in M$ such that $s \rightarrow t$ and $t \in \bar{\rho}(\varphi)$. In other words, $\bullet\varphi$ holds in the state s if there exists a next state t in which φ holds. We can define a dual construct $\circ\varphi \equiv \neg\bullet\neg\varphi$ called the "weak next", and show that $s \in \bar{\rho}(\circ\varphi)$ if and only if *for all* $t \in M$ such that $s \rightarrow t$, $t \in \bar{\rho}(\varphi)$. In other words, $\circ\varphi$ holds in the state s if for all next states t, φ holds in t. In particular, if s has no next state, $\circ\varphi$ holds in s ly. Figure 4 illustrates the meaning of $\bullet\varphi$ and $\circ\varphi$.

(Xiaohong). Add a diagram here showing why "strong next" $\bullet$ is interpreted as the "predecessor function" in models.

8.2. **Constructs defined using fixpoints.** "Strong next" $\bullet$ together with fixpoint constructs μ and ν provide great expressive power about transition systems. The following table summarizes a few important patterns and constructs about transition systems that are used later in the paper. Notice that the right column presents the *intended interpretation* of these patterns and constructs, where μ and ν are interpreted as true lfps and gfps.

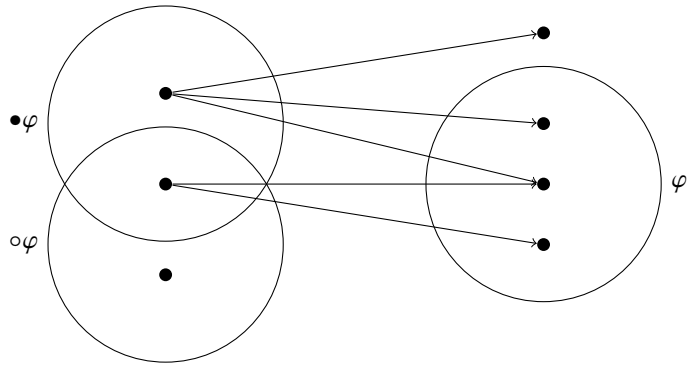

FIGURE 4. Strong next $\bullet\varphi$ and weak next $\circ\varphi$.

Matching logic patterns and constructs	Intended semantics in the transition system $\mathcal{M}$ Necessary and sufficient condition of $s \in \bar{\rho}(\text{lhs})$ for some state $s \in M$
$\bullet\varphi$	there exists a state t such that $s \to t$ and $t \in \bar{\rho}(\varphi)$
$\circ\varphi$	for all states t such that $s \to t$, $t \in \bar{\rho}(\varphi)$
$\bullet\top$	s is a non-terminating state (has a next state)
$\circ\bot$	s is a terminating state (has no next state)
$\mu f.\circ f$	all paths starting at s are finite
$\Diamond\varphi \equiv \mu f.(\varphi \vee \bullet f)$	there exists $n \geq 0$ and $t_1, \ldots, t_n$ such that $s \to t_1 \to \cdots \to t_n$ and $t_n \in \bar{\rho}(\varphi)$
$\Box\varphi \equiv \nu f.(\varphi \wedge \circ f)$	for all $n \geq 0$ and $t_1, \ldots, t_n$ such that $s \to t_1 \to \cdots \to t_n$, $t_n \in \bar{\rho}(\varphi)$
$\mu f.(\varphi_1 \vee (\varphi_2 \wedge \bullet f))$	there exists $n \geq 0$ and $t_1, \ldots, t_n$ such that $s \to t_1 \to \cdots \to t_n$, $t_n \in \bar{\rho}(\varphi_2)$ and $s, t_1, \ldots, t_{n-1} \in \bar{\rho}(\varphi_1)$
$\mu f.(\varphi_1 \vee (\varphi_2 \wedge \circ f))$	for all $n \geq 0$ and $t_1, \ldots, t_n$ such that $s \to t_1 \to \cdots \to t_n$, there exists $m \leq n$ such that $t_n \in \bar{\rho}(\varphi)$ $t_m \in \bar{\rho}(\varphi_2)$ and $s, t_1, \ldots, t_{m-1} \in \bar{\rho}(\varphi_1)$

Using these constructs, we can write axioms to specify properties of transition systems and rule out those which do not satisfy the properties. These axioms allow us to capture various interesting types of transition systems, which we summarize in the next table. Notice that we do not claim these axiomatizations are *complete*, because of the non-standard models of fixpoint constructs. However, we will show in the following subsections that these axioms *completely* capture various important logics for transition systems, including mu-calculus, linear temporal logic (LTL), computation tree logic (CTL), and CTL*.

Matching logic axioms	Transition system $\mathcal{M}$ in the intended semantics
(INF) $\bullet\top$	Every state in $\mathcal{M}$ has a next state
(FIN) $\mu f.\circ f$	$\mathcal{M}$ has no infinite trace
(LIN) $\bullet\varphi \to \circ\varphi$	Every state in $\mathcal{M}$, if it has next states, has a unique next state; In other words, every state has a linear future

Labeled transition systems can be captured in the similar way. Given a labeled transition system $\mathcal{M} = (M, A, \{\xrightarrow{a}\}_{a \in A})$. Define a matching logic signature $\mathbb{\Sigma}$ which contains a unary symbol $\bullet_a$ for every label $a \in A$.

9. INSTANCE: LINEAR TEMPORAL LOGIC

Linear temporal logic (LTL) is parametric on a set AP of atomic propositions. In literature, the term LTL often refers to the *infinite-trace LTL*, where LTL formulas are interpreted on infinite traces. In program verification and especially runtime verification [**?**],

finite execution traces play an important role, and thus *finite-trace LTL* is considered. In this section, we will show how both finite- and infinite-trace LTL can be defined in a uniform way in matching logic.

9.0.1. *Infinite-trace linear temporal logic.* Readers should be more familiar with infinite-trace LTL, so let us consider that first. The syntax of infinite-trace LTL extends the syntax of propositional calculus with a "next" modality $\circ$ and a "strong until" modality U_s:

$$\varphi ::= p \in \mathrm{AP} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \circ\varphi \mid \varphi \,\mathsf{U}_s\, \varphi$$

As usual, we define $\Diamond\varphi \equiv \mathit{true}\ \mathsf{U}_s\ \varphi$ and $\Box\varphi \equiv \neg(\Diamond\neg\varphi)$. Infinite-trace LTL formulas are interpreted on *infinite traces of sets of atomic propositions*, denoted as $\textsc{Traces}^\omega = [\mathbb{N} \to \mathcal{P}(\mathrm{AP})]$. We use $\alpha = \alpha_0\alpha_1\ldots$ to denote an infinite trace, and use the conventional notation $\alpha_{\geq i}$ to denote the suffix trace $\alpha_i\alpha_{i+1}\ldots$. Infinite-trace LTL semantics $\alpha \vDash_{\mathrm{infLTL}} \varphi$ is defined inductively as follows:

- $\alpha \vDash_{\mathrm{infLTL}} p$ if $p \in \alpha_0$ for atomic proposition p;
- $\alpha \vDash_{\mathrm{infLTL}} \varphi_1 \wedge \varphi_2$ if $\alpha \vDash_{\mathrm{infLTL}} \varphi_1$ and $\alpha \vDash \varphi_2$;
- $\alpha \vDash_{\mathrm{infLTL}} \neg\varphi$ if $\alpha \nvDash_{\mathrm{infLTL}} \varphi$;
- $\alpha \vDash_{\mathrm{infLTL}} \circ\varphi$ if $\alpha_{\geq 1} \vDash_{\mathrm{infLTL}} \varphi$;
- $\alpha \vDash_{\mathrm{infLTL}} \varphi_1 \,\mathsf{U}_s\, \varphi_2$ if there is $j \geq 0$ such that $\alpha_{\geq j} \vDash_{\mathrm{infLTL}} \varphi_2$ and for every $i < j$, $\alpha_{\geq i} \vDash_{\mathrm{infLTL}} \varphi_1$.

An infinite-trace LTL formula φ is valid, denoted as $\vDash_{\mathrm{infLTL}} \varphi$, if $\alpha \vDash_{\mathrm{infLTL}} \varphi$ for every $\alpha \in \textsc{Traces}^\omega$. A sound and complete proof system of infinite-trace LTL is given as follows. We write $\vdash_{\mathrm{infLTL}} \varphi$ if an infinite-trace LTL formula φ is provable.

Proof system of infinite-trace LTL extends propositional calculus with the following:

($\mathrm{K}_\circ$)	$\circ(\varphi_1 \to \varphi_2) \to (\circ\varphi_1 \to \circ\varphi_2)$	($\mathrm{N}_\circ$)	$\dfrac{\varphi}{\circ\varphi}$
($\mathrm{K}_\Box$)	$\Box(\varphi_1 \to \varphi_2) \to (\Box\varphi_1 \to \Box\varphi_2)$	($\mathrm{N}_\Box$)	$\dfrac{\varphi}{\Box\varphi}$
(Fun)	$\circ\varphi \leftrightarrow \neg(\circ\neg\varphi)$	(U_1)	$(\varphi_1 \,\mathsf{U}_s\, \varphi_2) \to \Diamond\varphi_2$
(U_2)	$(\varphi_1 \,\mathsf{U}_s\, \varphi_2) \leftrightarrow (\varphi_2 \vee (\varphi_1 \wedge \circ(\varphi_1 \,\mathsf{U}_s\, \varphi_2)))$	(Ind)	$\Box(\varphi \to \circ\varphi) \to (\varphi \to \Box\varphi)$

We can define a single-sorted matching logic theory $\mathsf{infLTL} = (\mathbb{\Sigma}, H)$ that faithfully captures infinite-trace LTL. The theory infLTL contains all definitions that are needed for the fixpoint constructs μ and ν. The signature Σ contains

- a unary symbol $\bullet$ called "strong-next"; We write $\bullet\varphi$ instead of $\bullet(\varphi)$;
- a constant symbol p for every atomic proposition $p \in \mathrm{AP}$.

We define the following derived matching logic constructs:

$$\circ\varphi \equiv \neg(\bullet\neg\varphi) \quad \Box\varphi \equiv \nu f.(\varphi \wedge \circ f) \quad \Diamond\varphi \equiv \mu f.(\varphi \vee \bullet f) \quad \varphi_1 \,\mathsf{U}_s\, \varphi_2 \equiv \mu f.(\varphi_2 \vee (\varphi_1 \wedge \bullet f))$$

In addition to axioms about fixpoint constructs, the axiom set H contains two more axioms

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

As we have seen in Section **??**, axiom (Lin) and (Inf) give us linear and infinite traces, respectively. Notice that $\Box\varphi$ is not defined as a matching logic symbol, but defined using the fixpoint construct ν. By simple matching logic reasoning, we can show that $\Box\varphi = \neg\Diamond\neg\varphi$ and $\Diamond\varphi = \top \,\mathsf{U}_s\, \varphi$, as expected. Thanks to the above definitions and notations,

Any infinite-trace linear temporal logic formula is a pattern in theory infLTL.

Figure 3,

As in Section 7, we show that $\vdash_{\text{infLTL}} \varphi$ implies $\mathsf{infLTL} \vdash \varphi$ using the proof-theoretic approach, and that $\mathsf{infLTL} \vDash \varphi$ implies $\vDash_{\text{infLTL}} \varphi$ using the model-theoretic approach, and let the completeness of both logics do the rest. infLTL also provable in theory infLTL. Here, we only show the model-theoretic part. We define a matching logic model of theory infLTL called the *standard model*, denoted as $\mathcal{M}$, whose carrier set is the set $\textsc{Traces}^\omega$ of all infinite traces. It adopts the intended interpretation for fixpoint constructs. Other symbols and derived constructs in infLTL are interpreted as follows:

- $p_{\mathcal{M}} = \{\alpha \mid p \in \alpha_0\}$ for every atomic proposition $p \in \text{AP}$;
- $\alpha \in \bullet_{\mathcal{M}}(\beta)$ if $\beta = \alpha_{\geq 1}$, i.e., β is the immediate surfix of α;

Notice that the standard model $\mathcal{M}$ satisfies (Lin) and (Fin), and thus is indeed a model of theory infLTL. Let ρ be any matching logic valuation. By structural induction on infinite-trace LTL formulas, one can show that for any infinite-trace LTL formula φ and an infinite trace α, $\alpha \vDash_{\text{infLTL}} \varphi$ if and only if $\alpha \in \bar{\rho}(\varphi)$. And thus $\vDash_{\text{infLTL}} \varphi$ if and only if $\mathcal{M} \vDash \varphi$. This finishes the reason diagram in Figure 3, and we conclude the conservative extension result for infinite-trace LTL.

Theorem 9.1 (Conservative Extension for Infinite-trace Linear Temporal Logic)**.** *Let φ be an infinite-trace LTL formula and* infLTL *be the matching logic theory for infinite-trace LTL. Then, $\vdash_{\text{infLTL}} \varphi$ if and only if* $\mathsf{infLTL} \vdash \varphi$.

9.0.2. *Finite-trace linear temporal logic.* Like infinite-trace LTL, finite-trace LTL formulas are interpreted on linear structures, i.e., traces. Unlike infinite-trace LTL, finite-trace LTL formulas are interpreted on finite traces. The syntax of finite-trace LTL is defined as follows:

$$\varphi ::= p \in \text{AP} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \circ\varphi \mid \varphi \mathsf{U}_w \varphi$$

As usual, define $\bullet\varphi \equiv \neg\circ\neg\varphi$, $\Box\varphi \equiv \varphi \mathsf{U}_w \mathit{false}$, and $\Diamond\varphi \equiv \neg(\Box\neg\varphi)$. Notice that we work with "weak until" U_w, different from infinite-trace LTL. As a result, in finite-trace LTL "always" $\Box$ is firstly defined, followed by "eventually" $\Diamond$. The two until's are different in that $\varphi_1 \mathsf{U}_s \varphi_2$ requires φ_2 eventually holds while $\varphi_1 \mathsf{U}_w \varphi_2$ does not. We refer readers to [**?**] for a more detailed discussion about finite-trace LTL and its relation with infinite-trace LTL.

Finite-trace LTL formulas are interpreted on nonempty finite traces of sets of atomic propositions, denoted as $\alpha = \alpha_0 \dots \alpha_n$. We write $\textsc{Traces}^*$ to denote the set of all finite traces. The semantics $\alpha \vDash_{\text{finLTL}} \varphi$ is defined similar to infinite-trace LTL. Notice $\circ\varphi$ holds in any singleton traces.

- $\alpha_0 \dots \alpha_n \vDash_{\text{finLTL}} p$ if $p \in \alpha_0$ for atomic proposition p;
- $\alpha_0 \dots \alpha_n \vDash_{\text{finLTL}} \varphi_1 \wedge \varphi_2$ if $\alpha_0 \dots \alpha_n \vDash_{\text{finLTL}} \varphi_1$ and $\alpha_0 \dots \alpha_n \vDash \varphi_2$;
- $\alpha_0 \dots \alpha_n \vDash_{\text{finLTL}} \neg\varphi$ if $\alpha_0 \dots \alpha_n \nvDash_{\text{finLTL}} \varphi$;
- $\alpha_0 \dots \alpha_n \vDash_{\text{finLTL}} \circ\varphi$ if $n = 0$ or $\alpha_1 \dots \alpha_n \vDash_{\text{finLTL}} \varphi$;
- $\alpha_0 \dots \alpha_n \vDash_{\text{finLTL}} \varphi_1 \mathsf{U}_w \varphi_2$ if either for every $i \leq n$, $s_i \dots s_n \vDash_{\text{finLTL}} \varphi_1$, or there is $j \leq n$ such that $\alpha_j \dots \alpha_n \vDash_{\text{finLTL}} \varphi_2$ and for every $i < j$, $\alpha_i \dots \alpha_n \vDash_{\text{finLTL}} \varphi_1$.

Finite-trace LTL has a sound and complete proof system.

Proof system of finite-trace LTL extends propositional calculus with the following:

($K_\circ$)	$\circ(\varphi_1 \to \varphi_2) \to (\circ\varphi_1 \to \circ\varphi_2)$	($N_\circ$)	$\dfrac{\varphi}{\circ\varphi}$
($K_\Box$)	$\Box(\varphi_1 \to \varphi_2) \to (\Box\varphi_1 \to \Box\varphi_2)$	($N_\Box$)	$\dfrac{\varphi}{\Box\varphi}$
($\neg\circ$)	$\neg\circ\varphi \to \circ\neg\varphi$	(coInd)	$\dfrac{\circ\varphi \to \varphi}{\varphi}$
(Fix)	$(\varphi_1 \mathsf{U}_w\ \varphi_2) \leftrightarrow (\varphi_2 \vee (\varphi_1 \wedge \circ(\varphi_1 \mathsf{U}_w\ \varphi_2)))$		

A matching logic theory $\mathsf{finLTL} = (\mathbb{\Sigma}, H)$ that captures finite-trace LTL can be defined similarly as theory infLTL, while instead of axiom (Inf), we add axiom (Fin) to capture the finite-trace semantics. A conservative extension result is proved in the same way; the standard model of theory finLTL has $\textsc{Traces}^\omega$ as its carrier set.

We point out that the defining proof rule (coInd) in finite-trace LTL is provable from (Fin). In fact, we have $\vdash \lfloor \circ\varphi \to \varphi \rfloor \to (\mu f.\circ f \to \varphi)$ by axiom (Lfp), and the rest is by simple matching logic reasoning.

We end this subsection by stating the conservative extension result for finite-trace LTL.

Theorem 9.2 (Conservative Extension for Finite-trace Linear Temporal Logic)**.** *Let φ be a finite-trace LTL formula and* finLTL *be the matching logic theory for finite-trace LTL defined as above. Then, $\vdash_{\text{finLTL}} \varphi$ if and only if* finLTL $\vdash \varphi$.

9.0.3. *Unifying infinite- and finite-trace linear temporal logics.* We propose a matching logic theory $\mathsf{LTL} = (\mathbb{\Sigma}, H)$ that unifies both infinite- and finite-trace LTLs. The signature $\mathbb{\Sigma}$ contains a unary symbol $\bullet$ for "strong next", and conventional temporal modalities are defined in their usual way. The axiom set H contains (Lin) to capture the "linear trace" semantics of both LTLs, and one can always add axioms, (Inf) or (Fin), to obtain infinite- or finite-trace LTL. Therefore, the matching logic theory LTL provides a unified, flexible, and extensible way to reason about properties about linear structures. Instead of designing new logics for every subclasses linear structures of interest, we can use a fixed logic (the matching logic), and write axioms to restrict models and structures. We will see more examples in the following subsections.

10. Instance: Computation Tree Logic

Computation tree logic (CTL) is another popular logic to reason about properties of transition systems. Unlike LTL, CTL is a *branching time* logic. It interprets its formulas on infinite trees and has modalities that can quantify paths in a tree. The syntax of CTL is parametric on a set AP of atomic propositions and is defined as follows.

$$\varphi ::= p \in \mathrm{AP} \mid \varphi \wedge \varphi \mid \neg\varphi \mid \mathsf{AX}\varphi \mid \mathsf{EX}\varphi \mid \varphi\ \mathsf{AU}\ \varphi \mid \varphi\ \mathsf{EU}\ \varphi$$

Other CTL modalities are defined in the usual way:

$$\mathsf{EF}\varphi \equiv \mathit{true}\ \mathsf{EU}\ \varphi \qquad \mathsf{AG}\varphi \equiv \neg\mathsf{EF}\neg\varphi \qquad \mathsf{AF}\varphi \equiv \mathit{true}\ \mathsf{AU}\ \varphi \qquad \mathsf{EG}\varphi \equiv \neg\mathsf{AG}\neg\varphi$$

Every CTL modality contains two letters; the first means either "all-path" A or "one-path" E, and the second means "next" X, "until" U, "always" G, or "eventually" F. Therefore, AX is "all-path next", and EU is "one-path until", etc.

Let $\textsc{Trees}^\omega$ be the set of all infinite trees over AP. An infinite tree τ has sets of AP as its nodes; it is *infinite* in the sense that it has no leaves and every node has children. We write $\mathsf{root}(\tau)$ to denote the root of τ. We write $\tau \to \tau'$ if τ' is an immediate subtree of τ. CTL semantics $\tau \vDash_{\text{CTL}} \varphi$ is defined inductively as follows.

- $\tau \vDash_{\text{CTL}} p$ if $p \in \mathsf{root}(\tau)$ for atomic proposition $p \in \text{AP}$;
- $\tau \vDash_{\text{CTL}} \varphi_1 \wedge \varphi_2$ if $\tau \vDash_{\text{CTL}} \varphi_1$ and $\tau \vDash_{\text{CTL}} \varphi_2$;
- $\tau \vDash_{\text{CTL}} \neg\varphi$ if $\tau \nvDash_{\text{CTL}} \varphi$;
- $\tau \vDash_{\text{CTL}} \mathsf{AX}\varphi$ if for all τ' such that $\tau \longrightarrow \tau'$, $\tau' \vDash_{\text{CTL}} \varphi$;
- $\tau \vDash_{\text{CTL}} \mathsf{EX}\varphi$ if there exists τ' such that $\tau \longrightarrow \tau'$ and $\tau' \vDash_{\text{CTL}} \varphi$;
- $\tau \vDash_{\text{CTL}} \varphi_1 \,\mathsf{AU}\, \varphi_2$ if for all $\tau_0, \tau_1, \ldots$ such that $\tau = \tau_0 \longrightarrow \tau_1 \longrightarrow \ldots$, there exists $i \geq 0$ such that $\tau_i \vDash_{\text{CTL}} \varphi_2$ and for all $j < i$, $\tau_j \vDash_{\text{CTL}} \varphi_1$;
- $\tau \vDash_{\text{CTL}} \varphi_1 \,\mathsf{EU}\, \varphi_2$ if there exists $\tau_0, \tau_1, \ldots$ such that $\tau = \tau_0 \longrightarrow \tau_1 \longrightarrow \ldots$, and there exists $i \geq 0$ such that $\tau_i \vDash_{\text{CTL}} \varphi_2$ and for all $j < i$, $\tau_j \vDash_{\text{CTL}} \varphi_1$;

We write $\vDash_{\text{CTL}} \varphi$ if $\tau \vDash_{\text{CTL}} \varphi$ for all τ. CTL admits a sound and complete proof system shown as follows. We write $\vdash_{\text{CTL}} \varphi$ if φ is provable in CTL.

Proof system of computational tree logic extends propositional calculus with the following:

(CTL_1)	$\mathsf{EX}(\varphi_1 \vee \varphi_2) \leftrightarrow \mathsf{EX}\varphi_1 \vee \mathsf{EX}\varphi_2$
(CTL_2)	$\mathsf{AX}\varphi \leftrightarrow \neg(\mathsf{EX}\neg\varphi)$
(CTL_3)	$\varphi_1 \,\mathsf{EU}\, \varphi_2 \leftrightarrow \varphi_2 \vee (\varphi_1 \wedge \mathsf{EX}(\varphi_1 \,\mathsf{EU}\, \varphi_2))$
(CTL_4)	$\varphi_1 \,\mathsf{AU}\, \varphi_2 \leftrightarrow \varphi_2 \vee (\varphi_1 \wedge \mathsf{AX}(\varphi_1 \,\mathsf{AU}\, \varphi_2))$
(CTL_5)	$\mathsf{EX}\mathit{true} \wedge \mathsf{AX}\mathit{true}$
(CTL_6)	$\mathsf{AG}(\varphi_3 \longrightarrow (\neg\varphi_2 \wedge \mathsf{EX}\varphi_3)) \longrightarrow (\varphi_3 \longrightarrow \neg(\varphi_1 \,\mathsf{AU}\, \varphi_2))$
(CTL_7)	$\mathsf{AG}(\varphi_3 \longrightarrow (\neg\varphi_2 \wedge (\varphi_1 \longrightarrow \mathsf{AX}\varphi_3))) \longrightarrow (\varphi_3 \longrightarrow \neg(\varphi_1 \,\mathsf{EU}\, \varphi_2))$
(CTL_8)	$\mathsf{AG}(\varphi_1 \longrightarrow \varphi_2) \longrightarrow (\mathsf{EX}\varphi_1 \longrightarrow \mathsf{EX}\varphi_2)$

We can define a matching logic theory $\mathsf{CTL} = (\Sigma, H)$ that faithfully captures CTL. The theory CTL contains all definitions that are needed for fixpoint constructs μ and ν and the unary symbol "strong next" $\bullet$. Define "weak next" $\circ\varphi \equiv \neg\bullet\neg\varphi$ as usual. In addition, the signature Σ contains a constant symbol p for every atomic proposition $p \in \text{AP}$. The axiom set H contains fixpoint axioms plus axiom (Inf) to capture the infinite tree semantics of CTL. In other words, remove axiom (Lin) from theory infLTL and we obtain the theory CTL. We define CTL modalities as derived constructs in matching logic as follows:

$$\mathsf{AX}\varphi \equiv \circ\varphi \quad \mathsf{EX}\varphi \equiv \bullet\varphi \quad \varphi_1 \,\mathsf{AU}\, \varphi_2 \equiv \mu f.\varphi_2 \vee (\varphi_1 \wedge \circ f) \quad \varphi_1 \,\mathsf{EU}\, \varphi_2 \equiv \mu f.\varphi_2 \vee (\varphi_1 \wedge \bullet f)$$

With the above definitions and notations,

Any computational tree logic formula is a matching logic pattern of theory CTL.

The standard model of theory CTL has the set Trees^ω of all infinite trees as its carrier set. It adopts intended semantics for fixpoint constructs μ and ν. Other symbols are interpreted as follows:

- $p_{\mathcal{M}} = \{\tau \mid p \in \mathsf{root}(\tau)\}$ for atomic proposition $p \in \text{AP}$;
- $\tau \in \bullet_{\mathcal{M}}(\tau')$ if $\tau \longrightarrow \tau'$;

One can prove that CTL validity coincides with the validity in this standard model. In addition, one can prove that all CTL proof rules and axioms are provable in matching logic. As in Section 9, this gives us the following conservative extension result for CTL.

Theorem 10.1 (Conservative Extension for Computational Tree Logic)**.** *Let φ be a computational tree logic formula and* CTL *be the matching logic theory for computational tree logic defined as above. Then, $\vdash_{CTL} \varphi$ if and only if* $\mathsf{CTL} \vdash \varphi$.

Before we end this subsection, we point out that matching logic provides a uniform way to study and play with variants of CTL. For example, CTL as presented here adopts infinite-tree semantics. One can consider a variant of CTL with finite-tree semantics, and it cannot be easier to do that in matching logic. One just needs to replace axiom (Inf) in

theory CTL with axiom (Fin), or simply remove it to capture both finite- and infinite CTLs. Instead of designing a new logic, one writes axioms to capture the intended semantics, and matching logic offers a sound and complete deduction for free. Even though the the axiomatization may not completely capture the intended semantics, it can faithfully and completely capture logics or calculi with complete deduction that are specifically designed for that semantics.

11. Instance: Propositional Dynamic Logic

Propositional dynamic logic (PDL) is an extension of modal logic to reason about programs. Its syntax is parametric on a set AP of atomic propositions and a set APgm of atomic programs. PDL has two types of formulas; *propositional formulas* are similar to formulas in modal logic or mu-calculus, and *program formulas (terms)* represent programs built from atomic programs and primitive regular expression operators.

$$\begin{array}{ll} \textit{propositional formulas} & \varphi ::= p \in \mathrm{AP} \mid \varphi \rightarrow \varphi \mid \textit{false} \mid [\alpha]\varphi \\ \textit{program formulas} & \alpha ::= a \in \mathrm{APgm} \mid \alpha\,;\alpha \mid \alpha \cup \alpha \mid \alpha^* \mid \alpha? \end{array}$$

Common propositional connectives can be defined from $\varphi \rightarrow \varphi$ and *false* in the usual way. Common program constructs such as if-then-else, while-do, and repeat-until statements can also be defined using the four primitive constructs. These are not our focus, so we refer readers to PDL literatures such as [**?**] for details. Define $\langle a\rangle\varphi \equiv \neg[\alpha](\neg\varphi)$ as in mu-calculus.

PDL formulas are interpreted on Kripke frames $\mathcal{M} = (M, [\![\_]\!]_{\mathcal{M}})$ where M is a state set and $[\![\_]\!]_{\mathcal{M}}$ is a meaning function that

- maps every atomic proposition $p \in \mathrm{AP}$ to a set of states $[\![p]\!]_{\mathcal{M}} \subseteq M$;
- maps every atomic programs $a \in \mathrm{APgm}$ to a binary relation on states $[\![a]\!]_{\mathcal{M}} \subseteq M \times M$.

Then, the meaning function is extended to all propositional and program formulasin the following mutual inductive way:

- $[\![\varphi_1 \rightarrow \varphi_2]\!]_{\mathcal{M}} = (M \setminus [\![\varphi_1]\!]_{\mathcal{M}}) \cup [\![\varphi_2]\!]_{\mathcal{M}}$;
- $[\![\textit{false}]\!]_{\mathcal{M}} = \emptyset$;
- $[\![[\alpha]\varphi]\!]_{\mathcal{M}} = \{s \mid \text{for all } t \in M \text{ such that } (s,t) \in [\![\alpha]\!]_{\mathcal{M}}, t \in [\![\varphi]\!]_{\mathcal{M}}\}$
- $[\![\alpha\,;\beta]\!]_{\mathcal{M}} = \{(s,t) \mid \text{ there exists a state } s' \text{ such that } (s,s') \in [\![\alpha]\!]_{\mathcal{M}} \text{ and } (s',t) \in [\![\beta]\!]_{\mathcal{M}}\}$
- $[\![\alpha \cup \beta]\!]_{\mathcal{M}} = [\![\alpha]\!]_{\mathcal{M}} \cup [\![\beta]\!]_{\mathcal{M}}$
- $[\![\alpha^*]\!]_{\mathcal{M}} = \bigcup_{n\geq 0} ([\![\alpha]\!]_{\mathcal{M}})^n$
- $[\![\varphi?]\!]_{\mathcal{M}} = [\![\varphi]\!]_{\mathcal{M}} \times [\![\varphi]\!]_{\mathcal{M}}$

A PDL formula φ is valid, denoted as $\vDash_{\mathrm{PDL}} \varphi$, if it holds in all Kripke frames. PDL has a sound and complete proof system. We write $\vdash_{\mathrm{PDL}} \varphi$ if φ is provable in PDL.

Proof system of propositional dynamic logic extends propositional calculus with the following:

(PDL_1)	$[\alpha](\varphi_1 \rightarrow \varphi_2) \rightarrow ([\alpha]\varphi_1 \rightarrow [\alpha]\varphi_2)$	(PDL_2)	$[\alpha](\varphi_1 \wedge \varphi_2) \leftrightarrow ([\alpha]\varphi_1 \wedge [\alpha]\varphi_2)$
(PDL_3)	$[\alpha \cup \beta]\varphi \leftrightarrow [\alpha]\varphi \wedge [\beta]\varphi$	(PDL_4)	$[\alpha\,;\beta]\varphi \leftrightarrow [\alpha][\beta]\varphi$
(PDL_5)	$[\psi?]\varphi \leftrightarrow (\psi \rightarrow \varphi)$	(PDL_6)	$\varphi \wedge [\alpha][\alpha^*]\varphi \leftrightarrow [\alpha^*]\varphi$
(PDL_7)	$\varphi \wedge [\alpha^*](\varphi \rightarrow [\alpha]\varphi) \rightarrow [\alpha^*]\varphi$	(Gen)	$\dfrac{\varphi}{[\alpha]\varphi}$

We compare PDL formulas $[\alpha]\varphi$ with mu-calculus formulas $[a]\varphi$. In mu-calculus, the action set is a discrete set and actions have no structure; while in PDL, programs have structures and are constructed from atomic ones withregular expression operators. Recall that in Section 7, we define a unary symbol a for every mu-calculus action a and define

$\langle a\rangle\varphi \equiv a(\varphi)$. This is known as a *shallow embedding*. In PDL, we adopt a different approach called *deep embedding*, which we elaborate in detail as follows.

We can define a matching logic theory $\mathsf{PDL} = (\mathbb{\Sigma}, H)$ that faithfully captures PDL. The signature $\mathbb{\Sigma} = (S, \Sigma)$ contains a sort set $S = \{state, pgm\}$ with a sort *state* for propositional formulas and a sort *pgm* for program formulas. The symbol set Σ contains

- a constant symbol $p \in \Sigma_{\lambda,state}$ for atomic proposition $p \in \mathrm{AP}$;
- a constant symbol $a \in \Sigma_{\lambda,pgm}$ for atomic program $a \in \mathrm{APGM}$;
- a unary symbol $\bullet \in \Sigma_{state,state}$ called "strong next";
- a binary symbol $\_;\_ \in \Sigma_{pgm\ pgm,pgm}$;
- a binary symbol $\_\cup\_ \in \Sigma_{pgm\ pgm,pgm}$;
- a unary symbol $\_^* \in \Sigma_{pgm,pgm}$;
- a unary symbol $\_? \in \Sigma_{state,pgm}$;

In addition, the theory PDL contains all definitions needed for fixpoint constructs. Define the notions $\langle\alpha\rangle\varphi \equiv \bullet(\alpha, \varphi)$ and $[\alpha]\varphi \equiv \neg\langle\alpha\rangle(\neg\varphi)$. With the above definitions and notations,

> *Any propositional formula of PDL is a matching logic pattern of sort state;*
> *Any program formula of PDL is a matching logic pattern of sort pgm;*

Theory PDL is called a deep embedding because it defines the concrete syntax of PDL programs in sort *pgm*, and has separate axioms defining their semantics. The axiom set H contains the following four defining axioms, one for each PDL program construct.

(Choice)	$[\alpha \cup \beta]\varphi = [\alpha]\varphi \wedge [\beta]\varphi$	(Seq)	$[\alpha\,;\beta]\varphi = [\alpha][\beta]\varphi$
(Test)	$[\psi?]\varphi = (\psi \rightarrow \varphi)$	(Iter)	$[\alpha^*]\varphi = \nu f.(\varphi \wedge [\alpha]f)$

Obviously, axioms (Choice), (Seq), and (Test) imply PDL axioms (PDL_3), (PDL_4), and (PDL_5), respectively. By easy fixpoint reasoning, one can prove that axiom (Iter) implies PDL axioms (PDL_6) and (PDL_7). In addition, (PDL_1), (PDL_2), and (Gen) are general properties about "strong next" $\bullet$ and also provable in theory PDL. Therefore, $\vdash_{\mathrm{PDL}} \varphi$ implies $\mathsf{PDL} \vdash \varphi$.

We claim that the four matching logic axioms for PDL program constructs defined in the above are more natural to understand and easier to design than the original PDL axioms. The PDL proof system is a blend of general modal logic reasoning, e.g., (PDL_1), (PDL_2), and (Gen), and specific axioms about program constructs, e.g, (PDL_3) – (PDL_7). Besides, axioms (PDL_6) are (PDL_7) are more like properties rather than a definition, because the formula $[\alpha^*]\varphi$ has multiple occurrences, while axiom (Iter) is clearly a definition.

One may argue that (Iter) uses the gfp construct ν and relies on axioms (Fix) and (Gfp), and (PDL_6) and (PDL_7) share the same style. We agree. And that is exactly why we think one should work in *a uniform and fixed logic* where general axioms about fixpoints can be defined. Then, we can use these axioms to reason about fixpoint properties and develop automatic tools and provers, rather than designing new logics and tools which all have their own ways to deal with fixpoints or induction axioms. We showed how mu-calculus, LTL, CTL, and PDL can be completely captured in matching logic with fixpoint axioms, and we hope it demonstrate that matching logic can be considered as a candidate of such a uniform and fixed logic.

We end this subsection with a conservative extension result for PDL. The result can be shown in the same way as in mu-calculus (see Figure 3), and we omit the proof.

Theorem 11.1 (Conservative Extension for Propositional Dynamic Logic)**.** *Let φ be a propositional formula in propositional dynamic logic and* PDL *is the matching logic for propositional dynamic logic defined as above. Then, $\vdash_{PDL} \varphi$ if and only if* $\mathsf{PDL} \vdash \varphi$.

12. Instance: Reachability Logic, (One-Path)

Reachability logic is a language-independent proof system for deriving reachability properties of systems and programs [**?**]. Its defining feature is the (Circularity) proof rule that supports reasoning about circular behavior of iterative and recursive program constructs. In historical literature [**?**], reachability logic is proposed alongside matching logic for program verification, where matching logic is used for defining static structure and program configurations, while reachability logic is used for reasoning about dynamic behavior.

In this section, we show that reachability logic is just an instance of matching μ-logic. What reachability logic offers is the power to specify and reason about dynamic properties about programs. In its essence, a program defines a transition system over *configurations*. We can very well achieve the same thing using matching μ-logic.

12.1. **Reachability logic basics.** Reachability logic is parametric on a matching logic model called the *underlying configuration model*, so before we introduce reachability logic syntax and semantics, let us first define the underlying configuration model.

Let $\mathbb{\Sigma}_{\text{cfg}}$ be a matching logic signature used to specify static program configurations. Depending on the target programming language of interest, the signature $\mathbb{\Sigma}_{\text{cfg}}$ may have various sorts and symbols, among which there is a distinguished sort *Cfg* for program configurations. Let M_{cfg} be a matching logic $\mathbb{\Sigma}_{\text{cfg}}$-model which we call the *underlying configuration model*. We write M_{Cfg} to mean the carrier set of sort *Cfg*, i.e., the set of all program configurations.

Reachability has simple syntax. The basic sentences are called *reachability rules*, which have the form

$$\varphi_1 \Rightarrow \varphi_2, \quad \varphi_1, \varphi_2 \text{ are patterns of sort } Cfg. \tag{30}$$

A *reachability system*, denoted as T, is a set of rules; it then yields a transition system over configurations, denoted as $\mathcal{T} = (M_{Cfg}, \rightarrow)$, where M_{Cfg} is the domain of all configurations in the underlying configuration model. The transition relation $\rightarrow$ is defined such that $s \rightarrow t$ if and only if there exists a rule $\varphi_1 \Rightarrow \varphi_2$ in T and a matching logic valuation ρ such that $s \in \bar{\rho}(\varphi_1)$ and $t \in \bar{\rho}(\varphi_2)$. Let $\rightarrow^* = \bigcup_{k \geq 0} (\rightarrow)^k$ be the transitive and reflexive closure of $\rightarrow$.

Let $\psi_1 \Rightarrow \psi_2$ be any reachability rule. We say it is ρ-valid, denoted as $T, \rho \vDash_{\text{uRL}} \psi_1 \Rightarrow \psi_2$, if for every configuration s such that $s \in \bar{\rho}(\varphi_1)$, either there is an infinite transition sequence $s \rightarrow s' \rightarrow s'' \rightarrow \ldots$ in the transition system $\mathcal{T}$ yielded by T, or there exists a configuration t such that $s \rightarrow^* t$ and $t \in \bar{\rho}(\varphi_2)$. Rule $\psi_1 \Rightarrow \psi_2$ is valid, denoted as $T \vDash_{\text{uRL}} \psi_1 \Rightarrow \psi_2$,if it is ρ-valid for every valuation ρ. Notice that validity in reachability logic is defined in the spirit of partial correctness.

reachability logic has a sound and relatively complete proof system as shown below. Notice the proof system derives more general sequents of the form $A \vdash_C \varphi_1 \Rightarrow \varphi_2$ where A and C are sets of rules. We call rules in A *axioms* and rules in C *circularities*. Rule $\varphi_1 \Rightarrow \varphi_2$ is provable in the reachability system T, denoted as $T \vdash_{\text{uRL}} \varphi_1 \Rightarrow \varphi_2$, if the sequent $T \vdash_\emptyset \varphi_1 \Rightarrow \varphi_2$ can be derived in the above proof system.

Theorem 12.1 (Soundness and completeness of reachability logic)**.** *Let $\mathbb{\Sigma}$ be a matching logic signature for configurations and $\mathcal{M}$ be the underlying configuration model. Let T be a reachability system and $\varphi_1 \Rightarrow \varphi_2$ be an reachability rule. Then $T \vDash_{uRL} \varphi_1 \Rightarrow \varphi_2$ if and only if $T \vdash_{uRL} \varphi_1 \Rightarrow \varphi_2$.*

The completeness of reachability logic is a *relative* one because the proof system consults the underlying configuration model $\mathcal{M}$ for validity in (Consequence) rule. In other words,

Proof system of reachability logic is parametric in a matching logic signature $\mathbb{\Sigma}$ for configurations and an underlying configuration model $\mathcal{M}$; it contains the following rules:

(Axiom) $\dfrac{\cdot}{A \vdash_C \varphi_1 \wedge \psi \Rightarrow \varphi_2 \wedge \psi}$ if $(\varphi_1 \Rightarrow \varphi_2) \in A$ and ψ is a predicate pattern

(Reflexivity) $\dfrac{\cdot}{A \vdash_\emptyset \varphi \Rightarrow \varphi}$ (Transitivity) $\dfrac{A \vdash_C \varphi_1 \Rightarrow \varphi_2 \quad A \cup C \vdash_\emptyset \varphi_2 \Rightarrow \varphi_3}{A \vdash_C \varphi_1 \Rightarrow \varphi_3}$

(Consequence) $\dfrac{\mathcal{M} \vDash \varphi_1 \rightarrow \varphi_1' \quad A \vdash_C \varphi_1' \Rightarrow \varphi_2' \quad \mathcal{M} \vDash \varphi_2' \rightarrow \varphi_2}{A \vdash_C \varphi_1 \Rightarrow \varphi_2}$

(Abstraction) $\dfrac{A \vdash_C \varphi_1 \Rightarrow \varphi_2}{A \vdash_C (\exists x.\varphi_1) \Rightarrow \varphi_2}$ if $x \notin FV(\varphi_2)$

(Circularity) $\dfrac{A \vdash_{C \cup \{\varphi_1 \Rightarrow \varphi_2\}} \varphi_1 \Rightarrow \varphi_2}{A \vdash_C \varphi_1 \Rightarrow \varphi_2}$ (Case Analysis) $\dfrac{A \vdash_C \varphi_1 \Rightarrow \varphi \quad A \vdash_C \varphi_2 \Rightarrow \varphi}{A \vdash_C \varphi_1 \vee \varphi_2 \Rightarrow \varphi}$

Figure 5. Reachability logic proof system

reachability logic is complete relative to the completeness of $\mathcal{M}$. If $\mathcal{M}$ can be completely axiomatized by a recursively enumerable set of axioms in matching logic, then $\mathcal{M} \vDash \varphi \rightarrow \varphi'$ is decidable, and the reachability logic (parametric on $\mathcal{M}$) becomes "absolutely" complete, i.e., there exists an algorithm that decides the validity of reachability rules. In practice, however, $\mathcal{M} \vDash \varphi \rightarrow \varphi'$ is often undecidable, and thus no algorithm can decide the validity of reachability rules. What Theorem 12.1 tells us is that this incompleteness origins in the undecidability of the underlying configuration model $\mathcal{M}$, not reachability logic itself.

12.2. **Reachability patterns.** Recall that $\mathbb{\Sigma}_{\text{cfg}}$ is the matching logic signature of configurations, and M_{cfg} is the underlying configuration model. An instance of reachability logic is then based on the underlying configuration model M_{cfg}.

Let us define an extended signature $\mathbb{\Sigma} = \mathbb{\Sigma}_{\text{cfg}} \cup \{\bullet \in \Sigma_{Cfg,Cfg}\}$ where $\bullet$ is the "strong next" symbol. The common temporal modalities in linear temporal logic and computation tree logic can be defined in the usual way. The objective here is to find a pattern that captures the reachability rule $\varphi_1 \Rightarrow \varphi_2$. Let T be a transition system where the reachability rule $\varphi_1 \Rightarrow \varphi_2$ holds. According to the semantics of reachability logic, this means that for every configuration $\gamma_0 \vDash \varphi_1$, there exists a complete path $\tau = \gamma_0\gamma_1\gamma_2\ldots$ such that τ is a finite path implies there exists $n \in \mathbb{N}$ such that $\gamma_n \vDash \varphi_2$. In other words, from configuration γ_0, either "eventually" φ_2 holds, in the usual LTL sense, or there exists an infinite path. This leads us to the following attempting definition of a reachability pattern:

$$\varphi_1 \Rightarrow \varphi_2 \equiv \varphi_1 \rightarrow ((\neg \mu X.\circ X) \vee \Diamond \varphi_2) \tag{31}$$

Recall that under standard semantics, the pattern $\mu X.\circ X$ is the set of states that are *well-founded*, meaning that they terminate on all paths. Therefore, $\neg \mu f.\circ f$ is exactly the set of states that admit an infinite path.

Proposition 12.2. *For any pattern φ and pattern set Γ, $\Gamma \vdash ((\neg \mu X.\circ X) \vee \Diamond \varphi) = \nu X.\varphi \vee \bullet \varphi$.*

Let us define "weak eventually" $\Diamond_w \varphi \equiv \nu X.\varphi \vee \bullet \varphi$.

12.3. **Reachability logic as an instance of matching μ-logic.** Reachability logic considers sequents that have the form $A \vdash_C \varphi_1 \Rightarrow \varphi_2$. In the following, we define a translation from reachability logic sequents to matching μ-logic obligations. This translation will be more complicated than the ones in LTL or CTL, so we define it gradually.

We first define the translation of reachability rules in the axiom set A. We denote this translation Axiom2MmL. According to the semantics of reachability logic, an axiom $\psi_1 \Rightarrow \psi_2 \in A$ means that the transition system *can make a step* from a configuration where ψ_1 holds to a configuration where ψ_2 holds. This leads to the following definition of the translation Axiom2MmL:

If the axiom set A is empty:

$\quad \text{Axiom2MmL}(A) = \top$;

If the axiom set $A = \{\psi_1 \Rightarrow \psi_2\} \cup A'$:

$\quad \text{Axiom2MmL}(A) = (\forall x_1 \ldots \forall x_n . (\psi_1 \rightarrow \bullet\psi_2)) \wedge \text{Axiom2MmL}(A')$

where $\{x_1, \ldots, x_n\} = FV(\psi_1) \cup FV(\psi_2)$

Here, the pattern $\psi_1 \rightarrow \bullet\psi_2$ captures the intuitive meaning that the transition system *can make a step* from any configurations that satisfies ψ_1 to some configurations where ψ_2 holds. The $\bullet$ symbol in $\bullet\psi_2$ makes sure the system indeed makes a step in order to satisfy $\Diamond_w\psi_2$. This becomes crucial when we prove all the proof rules in the reachability logic proof system in matching μ-logic, especially when we prove the (Circularity) rule. We point out that there are multiple ways to define the translation. For example, instead of $\psi_1 \rightarrow \bullet\Diamond_w\psi_2$, one can use $\psi_1 \rightarrow \bullet\psi_2$.

Next, let us define the translation of reachability rules in the circularity set C. We denote this translation Circ2MmL. The intuition of a reachability rule $\psi_1 \Rightarrow \psi_2$ in the circularity set is that the transition system will satisfy ψ_2 if it *makes any step* from a configuration that satisfy ψ_1. This leads to the following definition of the translation Circ2MmL:

If the circularity set C is empty:

$\quad \text{Circ2MmL}(C) = \top$;

If the circularity set $C = \{\psi_1 \Rightarrow \psi_2\} \cup C'$:

$\quad \text{Circ2MmL}(C) = (\forall x_1 \ldots \forall x_n . (\psi_1 \rightarrow \circ\Diamond_w\psi_2)) \wedge \text{Circ2MmL}(C')$

where $\{x_1, \ldots, x_n\} = FV(\psi_1) \cup FV(\psi_2)$

Here, the pattern $\psi_1 \rightarrow \circ\Diamond_w\psi_2$ captures the intuitive meaning that the transition system will eventually satisfy ψ_2 *after any steps* it makes at any configuration that satisfies ψ_1. The $\circ$ symbol in $\circ\Diamond_w\psi$ makes sure the system can satisfies $\Diamond_w\psi_2$ as long as it makes a step, and it does not matter what step it makes.

Finally, we define the translation of reachability rule $\varphi_1 \Rightarrow \varphi_2$ that appears in a reachability sequent, as in $A \vdash_C \varphi_1 \Rightarrow \varphi_2$. Depending on if the circularity set C is empty or not, the reachability rule $\varphi_1 \Rightarrow \varphi_2$ has different requirements on if φ_2 may hold at the current configuration or must hold sometime in the future. If C is empty, then φ_2 may hold at the current configuration; e.g., see (Reflexivity) rule. If C is not empty, then φ_2 can only hold after at least one step from the current configuration. This leads us to the next definition of transformation Rule2MmL, which takes a reachability rule $\varphi_1 \Rightarrow \varphi_2$ and a circularity set C as arguments:

If the circularity set C is empty:

$\quad \text{Rule2MmL}(\varphi_1 \Rightarrow \varphi_2, C) = \varphi_1 \rightarrow \Diamond_w\varphi_2$;

If the circularity set C is nonempty:

$\quad \text{Rule2MmL}(\varphi_1 \Rightarrow \varphi_2, C) = \varphi_1 \rightarrow \bullet\Diamond_w\varphi_2$.

In other words, when the circularity set C is not empty, the reachability sequent $A \vdash_C \varphi_1 \Rightarrow \varphi_2$ asks to prove something stronger than when the circularity set C is empty.

Now we are ready to transform any reachability logic sequence $A \vdash_C \varphi_1 \Rightarrow \varphi_2$ to the following proof obligation in matching μ-logic:

$$\vdash \text{Axiom2MmL}(A) \wedge \text{Circ2MmL}(C) \rightarrow \text{Rule2MmL}(\varphi_1 \Rightarrow \varphi_2, C)$$

Proposition 12.3. *Given any axiom set A and circularity set C of finitely many reachability rules, if the reachability logic sequent $A \vdash_C \varphi_1 \Rightarrow \varphi_2$ can be proved from the reachability logic proof system shown in Figure 5, then*

$$\text{Axiom2MmL}(A) \wedge \text{Circ2MmL}(C) \rightarrow \text{Rule2MmL}(\varphi_1 \Rightarrow \varphi_2, C)$$

is provable in matching μ-logic.

12.4. **Get Material From Here.**

Any reachability logic rule $\varphi_1 \Rightarrow \varphi_2$ is a matching logic pattern of sort Cfg.

Let uRL be a matching logic theory of signature $\mathbb{\Sigma}^{\rightarrow}$. We use it to capture reachability logic. The theory uRL contains all definitions needed for fixpoint constructs. In addition, it contains all valid patterns in the underlying configuration model $\mathcal{M}$ as axioms. This makes all implications needed for (Consequence) rule provable in theory uRL.

Let T be a reachability system and $\mathcal{T} = (M_{Cfg}, \rightarrow)$ be its yielded transition system. It is not hard to phrase the transition system $\mathcal{T}$ as a matching logic model of theory uRL and prove that $\mathcal{T} \models \varphi_1 \Rightarrow \varphi_2$ if and only if $T \vDash_{\text{uRL}} \varphi_1 \Rightarrow \varphi_2$. Extend theory uRL by adding all reachability rules in T as axioms and denote the extended theory as uRL_T. It follows that $\mathcal{T} \models \mathsf{uRL}_T$, and that $\mathsf{uRL}_T \models \varphi_1 \Rightarrow \varphi_2$ implies $T \vDash_{\text{uRL}} \varphi_1 \Rightarrow \varphi_2$.

We conjecture the following conservative extension theorem for reachability logic, whose proof is postponed to future work.

Conjecture 12.4 (Conservative Extension for Reachability Logic)**.** *Let $\mathbb{\Sigma}$ be a matching logic signature of configurations, $\mathcal{M}$ be an underlying configuration model, $\varphi_1 \Rightarrow \varphi_2$ be a reachability rule, T be a reachability system, and uRL_T be the corresponding matching logic theory all as defined above. Then, $T \vdash_{uRL} \varphi_1 \Rightarrow \varphi_2$ if and only if $\mathsf{uRL}_T \vdash \varphi_1 \Rightarrow \varphi_2$.*

To prove the conjecture, it suffices to prove that $T \vdash_{\text{uRL}} \varphi_1 \Rightarrow \varphi_2$ implies $\mathsf{uRL}_T \vdash \varphi_1 \Rightarrow \varphi_2$. In the following, we use an example originated from program verification problems to show how reachability proofs, especially application of (Circularity) rule, can be carried out in matching logic.

(Xiaohong). Finish the detail of example SUM

The invariant rule we want to prove is that

(Invariant) $$\exists n.(\varphi(n) \wedge n \geq 0) \Rightarrow \psi$$

Firstly, let us list some facts that are needed in the proof. Let us assume that the following two patterns are proved in advance.

(Base Case) $$\varphi(0) \rightarrow \Diamond\psi$$

(Loop Body) $$\varphi(n) \wedge n \geq 1 \rightarrow \bullet^k \varphi(n-1) \qquad \text{for some } k \geq 1$$

The following patterns are either valid in the underlying configuration model $\mathcal{M}$ or provable using fixpoint axioms.

(Domain)	$\exists n(\varphi(n) \wedge n \geq 0) = \varphi(0) \vee \exists n.(\varphi(n) \wedge n \geq 1)$
(Next Eventually)	$\Diamond\psi_1 = \bullet^k\psi_1$
(Comm)	$\forall n.\circ^k\psi_1 = \circ^k\forall n.\psi_1$
(Fix Next)	$\mu f.\circ f = \mu f.\circ^k f$
(Progress)	$\circ^k(\psi_1 \rightarrow \psi_2) \wedge \bullet^k\psi_1 \rightarrow \bullet^k\psi_2$

Finally we show the proof of (Invariant) in Figure 6. The symbol "$\Longleftarrow$" in the proof should be read as "to prove the above, it suffices to prove the below".

13. Instance: First-Order Logic with Least Fixpoints

$$\exists n.(\varphi(n) \wedge n \geq 0) \Rightarrow \psi$$

$\xleftarrow{\text{by definition}}$
$$\exists n.(\varphi(n) \wedge n \geq 0) \rightarrow (\mu f.\circ f \rightarrow \Diamond\psi)$$

$\xleftarrow{\text{by (Domain)}}$
$$\varphi(0) \vee \exists n.(\varphi(n) \wedge n \geq 1) \rightarrow (\mu f.\circ f \rightarrow \Diamond\psi)$$

$\xleftarrow{\text{by propositional reasoning}}$
$$(\varphi(0) \rightarrow (\mu f.\circ f \rightarrow \Diamond\psi)) \wedge (\exists n.(\varphi(n) \wedge n \geq 1) \rightarrow (\mu f.\circ f \rightarrow \Diamond\psi))$$

$\xleftarrow{\text{by (Base Case)}}$
$$\exists n.(\varphi(n) \wedge n \geq 1) \rightarrow (\mu f.\circ f \rightarrow \Diamond\psi)$$

$\xleftarrow{\text{by FOL reasoning}}$
$$\mu f.\circ f \rightarrow \forall n.(\varphi(n) \wedge n \geq 1 \rightarrow \Diamond\psi)$$

$\xleftarrow{\text{by (Fix Next)}}$
$$\mu f.\circ^k f \rightarrow \forall n.(\varphi(n) \wedge n \geq 1 \rightarrow \Diamond\psi)$$

$\xleftarrow{\text{by (Lfp)}}$
$$\circ^k \forall n.(\varphi(n) \wedge n \geq 1 \rightarrow \Diamond\psi) \rightarrow \forall n.(\varphi(n) \wedge n \geq 1 \rightarrow \Diamond\psi)$$

$\xleftarrow{\text{by (Comm)}}$
$$\forall n.\circ^k(\varphi(n) \wedge n \geq 1 \rightarrow \Diamond\psi) \rightarrow \forall n.(\varphi(n) \wedge n \geq 1 \rightarrow \Diamond\psi)$$

$\xleftarrow{\text{by FOL reasoning}}$
$$\circ^k(\varphi(n-1) \wedge n - 1 \geq 1 \rightarrow \Diamond\psi) \rightarrow (\varphi(n) \wedge n \geq 1) \rightarrow \Diamond\psi$$

$\xleftarrow{\text{by propositional reasoning}}$
$$\circ^k(\varphi(n-1) \wedge n - 1 \geq 1 \rightarrow \Diamond\psi) \wedge \varphi(n) \wedge n \geq 1 \rightarrow \Diamond\psi$$

$\xleftarrow{\text{by (Loop Body)}}$
$$\circ^k(\varphi(n-1) \wedge n - 1 \geq 1 \rightarrow \Diamond\psi) \wedge \bullet^k\varphi(n-1) \wedge n \geq 1 \rightarrow \Diamond\psi$$

$\xleftarrow{\text{by propositional reasoning}}$
$$\circ^k(\varphi(n-1) \wedge n \geq 2 \rightarrow \Diamond\psi) \wedge \bullet^k\varphi(n-1) \wedge n \geq 2 \rightarrow \Diamond\psi$$
$$\wedge \quad \circ^k(\varphi(n-1) \wedge n \geq 2 \rightarrow \Diamond\psi) \wedge \bullet^k\varphi(n-1) \wedge n = 1 \rightarrow \Diamond\psi$$

$\xleftarrow{\text{by propositional reasoning}}$
$$\circ^k(\varphi(n-1) \wedge n \geq 2 \rightarrow \Diamond\psi) \wedge \bullet^k\varphi(n-1) \wedge n \geq 2 \rightarrow \Diamond\psi$$
$$\wedge \quad \bullet^k\varphi(0) \rightarrow \Diamond\psi$$

$\xleftarrow{\text{by (Next Eventually)}}$
$$\circ^k(\varphi(n-1) \wedge n \geq 2 \rightarrow \Diamond\psi) \wedge \bullet^k\varphi(n-1) \wedge n \geq 2 \rightarrow \Diamond\psi$$
$$\wedge \quad \bullet^k\varphi(0) \rightarrow \bullet^k\Diamond\psi$$

$\xleftarrow{\text{by (Base Case) and frame reasoning}}$
$$\circ^k(\varphi(n-1) \wedge n \geq 2 \rightarrow \Diamond\psi) \wedge \bullet^k\varphi(n-1) \wedge n \geq 2 \rightarrow \Diamond\psi$$

$\xleftarrow{\text{by matching logic reasoning}}$
$$\circ^k(\varphi(n-1) \wedge n \geq 2 \rightarrow \Diamond\psi) \wedge \bullet^k(\varphi(n-1) \wedge n \geq 2) \rightarrow \Diamond\psi$$

$\xleftarrow{\text{by (Progress)}}$
$$\bullet^k(\Diamond\psi) \rightarrow \Diamond\psi$$

$\xleftarrow{\text{by (Next Eventually)}}$
QED

FIGURE 6. A proof of (Invariant).

14. INSTANCE: SEPARATION LOGIC

15. INSTANCE: LOGICS FOR SPECIFYING HYPERPROPERTIES

16. Instance: HyperLTL

HyperLTL is an extension of LTL for specifying *hyperproperties*, i.e., properties of systems. The *syntax* of HyperLTL is parametric on

- a countable set AP of *atomic propositions* denoted as a;
- an countably infinite set V of *path variables* denoted as $\pi_1, \pi_2, \ldots$.

HyperLTL *formulas* is defined by the following grammar:

$$\varphi ::= \forall \pi . \varphi \mid \exists \pi . \varphi \mid \psi \qquad \text{// quantifiers only appear at the top level}$$
$$\psi ::= a_\pi \mid \neg \psi \mid \psi \vee \psi \mid \circ \psi \mid \psi \; U \; \psi \qquad \text{// quantifier-free formulas}$$

An *infinite trace* $\tau \in (\mathcal{P}(\mathrm{AP}))^\omega$ is an infinite sequence of subsets of atomic propositions. In HyperLTL, we only consider infinite traces, so we abbreviate "infinite traces" as just "traces". We write $\tau[i]$ for mean the ith element of trace τ. The semantics of HyperLTL is defined w.r.t. a nonempty set of traces T and a valuation $\Pi : V \to T$ as follows:

- $T, \Pi, i \vDash_{\mathsf{hLTL}} a_\pi$ iff $a \in \Pi(\pi)[i]$;
- $T, \Pi, i \vDash_{\mathsf{hLTL}} \neg \psi$ iff $T, \Pi, i \nvDash_{\mathsf{hLTL}} \psi$;
- $T, \Pi, i \vDash_{\mathsf{hLTL}} \psi_1 \vee \psi_2$ iff $T, \Pi, i \vDash_{\mathsf{hLTL}} \psi_1$ or $T, \Pi, i \vDash_{\mathsf{hLTL}} \psi_2$;
- $T, \Pi, i \vDash_{\mathsf{hLTL}} \circ \psi$ iff $T, \Pi, i + 1 \vDash_{\mathsf{hLTL}} \psi$;
- $T, \Pi, i \vDash_{\mathsf{hLTL}} \psi_1 \; U \; \psi_2$ iff there exists $j \geq i$ such that $T, \Pi, j \vDash_{\mathsf{hLTL}} \psi_2$ and for all $i \leq k < j$ we have $T, \Pi, k \vDash_{\mathsf{hLTL}} \psi_1$;
- $T, \Pi, i \vDash_{\mathsf{hLTL}} \exists \pi . \varphi$ iff there exists $\tau \in T$ such that $T, \Pi[\tau/\pi], i \vDash_{\mathsf{hLTL}} \varphi$;
- $T, \Pi, i \vDash_{\mathsf{hLTL}} \forall \pi . \varphi$ iff for all $\tau \in T$ we have $T, \Pi[\tau/\pi], i \vDash_{\mathsf{hLTL}} \varphi$;

where $\Pi[\tau/\pi]$ denotes the valuation Π' such that $\Pi'(\pi) = \tau$ and $\Pi'(\pi') = \Pi(\pi)$ for all $\pi' \not\equiv \pi$.

16.1. **Defining HyperLTL in ML.** We define the signature $\mathbb{\Sigma}^{\mathsf{hLTL}} = (\{State\}, \mathrm{Var}, \Sigma^{\mathsf{hLTL}})$ such that Var contains all path variables in V as *element variables*, and Σ^{hLTL} contains the following symbols:

"one-path next"	$\bullet \in \Sigma^{\mathsf{hLTL}}_{State,State}$	"initial states"	$init \in \Sigma^{\mathsf{hLTL}}_{\lambda,State}$
"atomic proposition"	$a \in \Sigma^{\mathsf{hLTL}}_{\lambda,State}$	"successor states"	$succ \in \Sigma^{\mathsf{hLTL}}_{State,State}$
"box"	$box \in \Sigma^{\mathsf{hLTL}}_{State,State}$	"column"	$col \in \Sigma^{\mathsf{hLTL}}_{State,State}$

We assume all syntactic sugar that is defined for LTL. In particular, we have "all-path" next $\circ\varphi \equiv \neg\bullet\neg\varphi$ and "eventually" $\Diamond\varphi \equiv \mu X . \varphi \vee \bullet X$. We define the $\mathbb{\Sigma}^{\mathsf{hLTL}}$-theory Γ^{hLTL} that contains the following axioms:

(Inf) $\bullet\top$ (Succ) $succ(x) = \exists y . y \wedge (x \in \bullet y)$

(Lin) $\circ\varphi \to \bullet\varphi$ (Box) $box(x) = \mu X . init \vee (succ(X) \wedge \lceil \Diamond x \wedge succ(X) \rceil)$

(Column) $col(x) = box(x) \wedge \neg box(\bullet x)$

The pattern $succ(x)$ is the inverse of $\bullet$ and gives the set of successors of x. An intuitive illustration of $box(x)$ and $col(x)$ will be given later.

Lemma 16.1. (Lin) *implies that succ is a partial function, i.e.,* $\vdash \exists y . succ(x) \subseteq y$. (Lin) *and* (Inf) *imply that succ is a total function, i.e.,* $\vdash \exists y . succ(x) = y$.

To capture HyperLTL syntax, we define the following syntactic sugar:

$$a_\pi \equiv col(a \wedge \mu X . \pi \vee succ(X)) \quad \forall \pi . \varphi \equiv \forall \pi . \pi \in init \to \varphi \quad \exists \pi . \varphi \equiv \exists \pi . \pi \in init \wedge \varphi$$

With the above syntactic sugar, all HyperLTL formulas are $\mathbb{\Sigma}^{\mathsf{hLTL}}$-patterns of sort $State$.

Lemma 16.2. *$T, \Pi, i \vDash_{\mathsf{hLTL}} \varphi$ iff $[\![i]\!] \subseteq \overline{\rho_\Pi}(\varphi)$ and $T, \Pi, i \nvDash_{\mathsf{hLTL}} \varphi$ iff $[\![i]\!] \cap \overline{\rho_\Pi}(\varphi) = \emptyset$.*

17. Get Material From Here

Intuitively, $\bar{\rho}(\varphi)$ is the set of elements that match the pattern φ. Derived constructs are defined as follows for convenience:

$$\begin{array}{rclcrcl} \top_s & \equiv & \exists x{:}s.x{:}s & & \bot_s & \equiv & \neg\top_s \\ \varphi_1 \vee \varphi_2 & \equiv & \neg(\neg\varphi_1 \wedge \neg\varphi_2) & & \varphi_1 \rightarrow \varphi_2 & \equiv & \neg\varphi_1 \vee \varphi_2 \\ \varphi_1 \leftrightarrow \varphi_2 & \equiv & (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1) & & \forall x.\varphi & \equiv & \neg(\exists x.\neg\varphi) \end{array}$$

Interested readers are encouraged to prove these derived constructs have the intended semantics, or refer to [**?**] for details. We often drop the sort subscripts when there is no confusion.

Given a model $\mathcal{M}$ and a valuation ρ, we say $\mathcal{M}$ and ρ satisfy a pattern φ_s, denoted as $\mathcal{M}, \rho \vDash \varphi_s$, if $\bar{\rho}(\varphi) = M_s$. We say $\mathcal{M}$ satisfies φ_s or φ_s holds in $\mathcal{M}$, denoted as $\mathcal{M} \vDash \varphi_s$, if $\mathcal{M}, \rho \vDash \varphi_s$ for every valuation ρ. We say φ_s is valid if it holds in every model. Let Γ be a pattern set. We say $\mathcal{M}$ satisfies Γ, if $\mathcal{M} \vDash \varphi$ for every $\varphi \in \Gamma$. We say Γ semantically entails φ, denoted as $\Gamma \vDash \varphi$, if for every model $\mathcal{M}$ such that $\mathcal{M} \vDash \Gamma$, $\mathcal{M} \vDash \varphi$. When Γ is the empty set, we abbreviate $\emptyset \vDash \varphi$ as just $\vDash \varphi$, which is equivalent to say φ is valid.

Given a signature $\mathbb{\Sigma}$, matching logic gives us all $\mathbb{\Sigma}$-models. Sometimes, we are only interested in some models, instead of all models. There are typically two ways to restrict models. One way is to define a syntactic matching logic theory $(\mathbb{\Sigma}, H)$ where H is a set of patterns called axioms. A model $\mathcal{M}$ belongs to the theory $(\mathbb{\Sigma}, H)$ if $M \vDash H$. Syntactic theories are preferred if the models of interest can be axiomatized by a recursively enumerable set H. Most syntactic theories defined in this paper have finite axiom sets. Alternatively, we can define a semantic matching logic theory $(\mathbb{\Sigma}, C)$ where C is a collection of $\mathbb{\Sigma}$-models. A model $\mathcal{M}$ belongs to the theory $(\mathbb{\Sigma}, C)$ if $\mathcal{M} \vDash \varphi$ for all φ that holds in all models in C. Usually, the collection C is a singleton set containing exactly one model $\mathcal{I}$, often referred as the intended model or the standard model, and we abbreviate $(\mathbb{\Sigma}, \{\mathcal{I}\})$ as $(\mathbb{\Sigma}, \mathcal{I})$. Semantic theories are preferred if the intended model is not axiomatized by any recursively enumerable set of axioms; this is often the case for initial algebra semantics or domains which are defined by induction, such as natural numbers (with multiplication) and finite maps. We will see an example of semantic theories in Section 17.1.4.

Syntactic theories support a notion of proofs (see Section **??**) as they have an axiom set, but semantic theories offer arbitrary expressiveness. They both are means to restricting models, and we simply say "theories" when their distinction is not the emphasis.

(Grigore). I think it is also important to state that when all models are assumed, then ML has been shown to have the same expressiveness as FOL=, but like it is the case with capturing SL, or in FOL with induction, or in initial algebra semantics, one can reduce the set of models and thus get arbitrary expressiveness.

17.1. **Known results about matching logic expressiveness power.** A few important logics and calculus are shown to be definable in matching logic, including propositional calculus, predicate logic, algebraic specification, first-order logic with equality, modal logic S5, and separation logic. In this section, we only discuss a few of them and refer interested readers to [**?**] for more details.

17.1.1. *Definedness, equality, membership, functions, and partial functions.* As we have seen, patterns are interpreted as sets. When it comes to classical reasoning for existing mathematical domains, we need a way to interpret patterns in a conventional, two-value

way; for example, the total set means true and the empty set means false. We also want to lift reasoning with sort s_1 to sort s_2. In matching logic, the above is methodologically achieved by *definedness* symbols. For any two sorts s and s' which need not be distinct, the definedness symbol $\lceil\_\rceil_s^{s'} \in \Sigma_{s,s'}$ is a unary matching logic symbol which has an axiom $\lceil x{:}s\rceil_s^{s'}$ called the definedness axiom. The axioms makes $\lceil\_\rceil_s^{s'}$ behaves like a predicate that checks definedness:

$$\bar{\rho}(\lceil\varphi_s\rceil_s^{s'}) = M_{s'} \quad \text{if } \bar{\rho}(\varphi_s) \neq \emptyset \qquad\qquad \bar{\rho}(\lceil\varphi_s\rceil_s^{s'}) = \emptyset \quad \text{if } \bar{\rho}(\varphi_s) = \emptyset$$

Definedness symbols allow us to define many useful derived constructs, including

$\lfloor\varphi\rfloor_s^{s'} \equiv \neg\lceil\neg\varphi\rceil_s^{s'}$ that checks totality $\quad x \in_s^{s'} \varphi \equiv \lceil x \wedge \varphi\rceil_s^{s'}$ that checks membership

$\varphi_1 =_s^{s'} \varphi_2 \equiv \lfloor\varphi_1 \leftrightarrow \varphi_2\rfloor_s^{s'}$ that checks equality $\quad \varphi_1 \subseteq_s^{s'} \varphi_2 \equiv \lfloor\varphi_1 \rightarrow \varphi_2\rfloor_s^{s'}$ that checks containment

These derived constructs have the intended semantics. For example,

$$\bar{\rho}(\varphi_1 =_s^{s'} \varphi_2) = M_{s'} \quad \text{if } \bar{\rho}(\varphi_1) = \bar{\rho}(\varphi_2) \qquad \bar{\rho}(\varphi_1 =_s^{s'} \varphi_2) = \emptyset \quad \text{if } \bar{\rho}(\varphi_1) \neq \bar{\rho}(\varphi_2)$$

We drop their sort subscripts if there is no confusion.

In matching logic, symbols are interpreted relationally $\sigma_M : M_{s_1}\times,\ldots,\times M_{s_n} \rightarrow 2^{M_s}$. Sometimes, we want to state a symbol σ is to be interpreted as a function in all models. This is achieved by adding the axiom $\exists y.\sigma(x_1,\ldots,x_n) = y$ where $x_1,\ldots,x_n,y$ are distinct. Partial functions can be defined in the similar way, by adding the axiom $\neg\sigma(x_1,\ldots,x_n) \vee \exists y.\sigma(x_1,\ldots,x_n) = y$. We point out that partial functions has been the main subject of research in partial first-order logic, with various logics and axioms proposed to capture the desired properties of definedness and undefinedness; while matching logic allows us elegantly define definedness and partial functions, without a need to develop a new logic.

17.1.2. *Modal logic S5.* One of the most popular modal logics, S5, is definable in matching logic. S5 is parametric on a set of atomic propositions AP, and its syntax, as shown below, extends propositional calculus with a unary modality$\Box$ that captures necessity: $\Box\varphi$ is read as "it is necessary that φ". A dual modality $\Diamond\varphi \equiv \neg(\Box\neg\varphi)$ is defined to capture possibility.

$$\varphi ::= \text{AP} \mid \varphi \rightarrow \varphi \mid \neg\varphi \mid \Box\varphi$$

S5 formulas are interpreted on a set W of worlds with a valuation $v : \text{AP}\times W \rightarrow \{true, false\}$ stating that each proposition holds in a given subset of worlds. The valuation is then extended to S5 formulas inductively:

- $v(\neg\varphi, w) = true$ if $v(\varphi, w) = false$; otherwise, $v(\neg\varphi, w) = false$.
- $v(\varphi_1 \rightarrow \varphi_2, w) = true$ if $v(\varphi_1, w) = false$ or $v(\varphi_2, w) = true$; otherwise, $v(\varphi_1 \rightarrow \varphi_2, w) = false$.
- $v(\Box\varphi, w) = true$ if $v(\varphi, w') = true$ for every $w' \in W$; otherwise, $v(\Box\varphi, w) = false$.

An S5-formula is valid, denoted as $\models_{S5} \varphi$, if for every world set W and valuation v, $v(\varphi, w) = true$ for every $w \in W$. S5 admits the following sound and complete proof system which can derive all valid S5-formulas:

Proof system of Modal logic S5 extends propositional calculus proof system with the following:

(N) $\dfrac{\varphi}{\Box\varphi}$ (K) $\Box(\varphi_1 \rightarrow \varphi_2) \rightarrow (\Box\varphi_1 \rightarrow \Box\varphi_2)$

(M) $\Box\varphi \rightarrow \varphi$ (5) $\Diamond\varphi \rightarrow \Box\Diamond\varphi$

Modal logic S5 can be faithfully captured by a matching logic theory which we refer to as S5. The theory S5 has a signature $\mathbb{\Sigma} = (S, \Sigma)$ where S contains exactly one sort, say *world*, and Σ contains a unary symbol $\Diamond \in \Sigma_{world,world}$, plus a constant symbol $p \in \Sigma_{\lambda,world}$

for every atomic proposition $p \in \mathrm{AP}$. The theory S5 has only one axiom, $\Diamond w$, which states that the symbol $\Diamond$ is the definedness symbol. The totality $\Box\varphi \equiv \neg(\Diamond\neg\varphi)$ is defined as a derived construct. With the above definitions,

Any S5-formula is a matching logic pattern in the theory S5.

We establish an important result known as the "conservative extension", which is proved via a model-theoretic approach. We elaborate the proof of conservative extension for S5 here as an example, since the same proof idea will show up multiple times in the rest of the paper. In short, we will show there is a one-to-one correspondence between pairs (W, v) of an S5 world set and a valuation, and matching logic models of the theory S5. The correspondence works in two directions. For the direction from S5 to matching logic, the correspondence takes W as the carrier set of sort *world*, and interprets $\Diamond$ as the definedness predicate $\Diamond_{world}(w) = W$ for every $w \in W$. Moreover, every atomic proposition $p \in \mathrm{AP}$ is interpreted to the subset of worlds $p_{world} = \{w \in W \mid v(p, w) = true\}$. The other direction from matching logic to S5 is left for the interested readers. Under this one-to-one correspondence, we can prove by structural induction that for every S5-formula φ,

$$v(\varphi, w) = true \quad \text{if and only if} \quad w \in \bar{\rho}(\varphi).$$

Notice that S5-formulas contain no variables as matching logic patterns, so it does not matter which valuation ρ we pick in the above. Immediately, we know $\vDash_{S5} \varphi$ if and only if $\mathsf{S5} \vDash \varphi$ for every S5-formula φ. By the completeness results of both S5 and matching logic, we know $\vdash_{S5} \varphi$ if and only if $\mathsf{S5} \vdash \varphi$ for every S5-formula φ.

17.1.3. *First-order logic and its variants.* TBC.

(Xiaohong). This section should say that FOL= and ML have the same expressiveness when considering all models. We may also show the reduction to FOL without equality, or why ML is more expressive than FOL without equality. We may show FOL+lfp here, too.

17.1.4. *Separation logic.* Separation logic is a logic specifically designed for reasoning about heap structures. The syntax of separation logic extends first-order logic with some heap operations:

$$\varphi ::= (\text{first-order logic syntax}) \mid emp \mid Nat \mapsto Nat \mid \varphi * \varphi \mid \varphi -\!\!* \varphi$$

Let $s\colon \textsc{Var} \rightharpoonup Nat$ be a partial function called a store and $h\colon Nat \rightharpoonup Nat$ be a partial function called a heap. Separation logic formulas are interpreted in the pair (s, h) inductively as follows:

- $(s, h) \vDash_{\mathrm{SL}} \varphi$ if $s \vDash_{\mathrm{FOL}} \varphi$ and φ is a first-order logic formula;
- $(s, h) \vDash_{\mathrm{SL}} emp$ if the domain of h is empty;
- $(s, h) \vDash_{\mathrm{SL}} e_1 \mapsto e_2$ if $\bar{s}(e_1) \neq 0$, the domain of h is $\{\bar{s}(e_1)\}$, and $h(\bar{s}(e_1)) = \bar{s}(e_2)$;
- $(s, h) \vDash_{\mathrm{SL}} \varphi_1 * \varphi_2$ if there exist disjoint h_1 and h_2 such that $h = h_1 * h_2$ and $(s, h_1) \vDash_{\mathrm{SL}} \varphi_1$ and $(s, h_2) \vDash_{\mathrm{SL}} \varphi_2$;
- $(s, h) \vDash_{\mathrm{SL}} \varphi_1 -\!\!* \varphi_2$ if for every h_1 disjoint with h, if $(s, h_1) \vDash_{\mathrm{SL}} \varphi_1$ then $(s, h_1 * h) \vDash_{\mathrm{SL}} \varphi_2$.

A separation logic formula φ is valid, denoted as $\vDash_{\mathrm{SL}} \varphi$, if $(s, h) \vDash_{\mathrm{SL}} \varphi$ for every store s and heap h.

Separation logic is faithfully captured by a matching logic theory we which denote as SL. The theory SL has the signature $\mathbb{\Sigma} = (S, \Sigma)$ where S contains a sort *Nat* for natural numbers and a sort *Map* for heaps. The symbol set Σ contains a constant symbol $emp \in \Sigma_{\lambda, Map}$, a

binary symbol $* \in \Sigma_{Nat\ Nat,Map}$, and a binary symbol $\mapsto \in \Sigma_{Map\ Map,Map}$. We write $*$ and $\mapsto$ in mixfix form. The separation conjunction is defined as an alias

$$\varphi_1 \mathrel{-\!\!*} \varphi_2 \equiv \exists h.(h \wedge \lfloor h * \varphi_1 \rightarrow \varphi_2 \rfloor)$$

With the above definitions,

Any separation logic formula is a matching logic pattern of sort Map in the theory SL.

Unlike in modal logic S5 where the theory S5 is defined with a set of axioms, the theory SL is defined by a particular matching logic model of the above signature known as the intended model or the standard model, denoted as *Map*. The intended model *Map* has the set of natural numbers $\mathbb{N}$ as the carrier set of sort *Nat*, and the set of partial functions $\{h \mid h\colon \mathbb{N} \rightharpoonup \mathbb{N}\}$ as the carrier set of sort *Map*. Symbols *emp*, $*$, and $\mapsto$ are interpreted in *Map* in the intended way. Notice that separation logic formulas contain no free variables of sort *Map* as matching logic patterns, so the valuation of variables of sort *Map* does not matter. In addition, there is a one-to-one correspondence between Valuations of variables of sort *Nat* and separation logic states. Under this correspondence, we can prove by structural induction that

$(s, h) \models_{\mathrm{SL}} \varphi$ if and only if $h \in \bar{\rho}(\varphi)$ // s and ρ conform to the one-to-one correspondence

As a corollary, $\models_{\mathrm{SL}} \varphi$ if and only if $\mathsf{SL} \models \varphi$ for every separation logic formula φ.