Chapter 3

Basics of Matching μ-Logic

Contents

The basic ingredients of matching μ-logic together with simple and natural examples are introduced in this chapter. The first section presents the syntax of matching μ-logic, emphasizing the two variants that are currently in use, one with an unsorted universe and the other with a many-sorted universe. The second section introduces the models of matching μ-logic, formalizing the notions of validity, satisfaction, and theories. The third and the last section are dedicated to the proof system of matching μ-logic.

3.1 Matching μ-Logic Syntax

A key concept in matching μ-logic is that of *patterns*, which are used to uniformly represent data structures, dynamic behaviors (of programs), and logical/mathematical constraints. Patterns are formally defined as follows.

Definition 3.1 (Matching Logic Patterns)**.** Given two disjoint sets $\mathbb{EV}$ and $\mathbb{SV}$ of *element variables* and *set variables*, respectively, and a signature Σ of *constant symbols*, the set of Σ-*patterns*, written $\mathbb{Pattern}(\Sigma)$, is inductively defined by the following grammar:

$\varphi ::= x$	Element Variables in $\mathbb{EV}$
$\mid X$	Set Variables in $\mathbb{SV}$
$\mid \sigma$	Symbols in Σ
$\mid \varphi_1\, \varphi_2$	Applications
$\mid \bot$	Bottom (Logical Fallacy)
$\mid \varphi_1 \to \varphi_2$	Implication
$\mid \exists x \,.\, \varphi$	Existential Quantification
$\mid \mu X \,.\, \varphi$	Least Fixpoints

where in $\mu X \,.\, \varphi$ we require that φ is positive in X, that is, X does not occur in an odd number of the left-hand sides of implication patterns.

Definition 3.1 has only 8 syntactic constructs, including variables, symbols, application, logical connectives, quantification, and fixpoints. Compared to first-order logic (FOL), matching μ-logic does not distinguish between terms and formulas. Instead, they are unified as patterns.

Intuitively, patterns can be matched by zero, one, or more elements. For example, an element variable x is matched by exactly one element, which is the one that x is assigned to. The disjunction $x \vee y$ (which can be defined from $\bot$ and $\to$ in the usual way; see Definition 3.15) is matched by either x or y. Bottom, written $\bot$, represents the logical fallacy, and is matched by no elements. Top, written $\top$, can be defined from $\bot$ and is matched by all elements.

Generally speaking, matching μ-logic builds patterns to match data structures that are of certain forms, or dynamic behaviors that satisfy certain properties. We use a series of simple and natural examples to illustrate the use of patterns.

Example 3.2. Let $\Sigma = \{\mathsf{zero}, \mathsf{succ}\}$, where zero represents the natural number 0 and succ represents the successor function. Then, the application pattern $\mathsf{succ}\ \mathsf{zero}$ represents the result of applying succ to zero, which is 1. The application $\mathsf{succ}\ (\mathsf{succ}\ \mathsf{zero})$ represents 2, etc. Note that both zero and succ are constant symbols, and we can use the binary application construct to build terms as in FOL.

Example 3.3. Let $\Sigma = \{\mathsf{zero}, \mathsf{succ}, \mathsf{plus}\}$, where plus represents the addition function. Then, the application pattern $(\mathsf{plus}\ \mathsf{zero})\ \mathsf{zero}$ represents $0+0$. In other words, we can build terms with many arguments by simply currying the application. For notational simplicity, we assume that application is *left associative*. Therefore, we can abbreviate $(\mathsf{plus}\ \mathsf{zero})\ \mathsf{zero}$ as $\mathsf{plus}\ \mathsf{zero}\ \mathsf{zero}$.

Example 3.4. A pattern can be matched by more than one elements. Let $\Sigma = \{\mathsf{zero}, \mathsf{succ}, \mathsf{plus}, \mathsf{even}\}$, where even is matched by all even numbers, including 0. Then, $\mathsf{succ}\ \mathsf{even}$ is matched by all odd numbers, and $\mathsf{succ}\ (\mathsf{succ}\ \mathsf{even})$ is matched by all even numbers greater than 0.

Example 3.5. Logical connectives can be regarded as set operations. Let $\Sigma = \{\mathsf{zero}, \mathsf{succ}, \mathsf{plus}, \mathsf{even}, \mathsf{nat3}\}$, where $\mathsf{nat3}$ is matched by all the multiples of 3. Then, $\mathsf{even} \wedge \mathsf{nat3}$ is matched by all even numbers that are multiples of 3, i.e., all the multiples of 6.

Example 3.6. Patterns can be matched by data structures. Let $\Sigma = \{\mathsf{emp}, \mathsf{mapsto}, \mathsf{merge}\}$, where emp represents the empty heap, mapsto is the constructor of singleton heaps, and merge represents the disjoint union of two heaps (also known as the "magic wand" in separation logic [2]). Then, $\mathsf{merge}\ (\mathsf{mapsto}\ 1\ 2)\ (\mathsf{mapsto}\ 3\ 4)$ is matched by the heap where location 1 has value 2, location 3 has value 4, and the other locations have no values in them. For notational simplicity, we define the following *notations*:

$$1 \mapsto 2 \equiv \mathsf{mapsto}\ 1\ 2$$
$$3 \mapsto 4 \equiv \mathsf{mapsto}\ 3\ 4$$
$$1 \mapsto 2 * 3 \mapsto 4 \equiv \mathsf{merge}\ (\mathsf{mapsto}\ 1\ 2)\ (\mathsf{mapsto}\ 3\ 4)$$

This way, the heap formulas in separation logic *are* matching μ-logic patterns *as is*.

Example 3.7. The quantifier $\exists$ can be used to create *abstraction*. Let $\Sigma = \{\mathsf{emp}, \mathsf{mapsto}, \mathsf{merge}\}$ be the same as in Example 3.6. Then, $1 \mapsto x$ (which is an abbreviation of the pattern $\mathsf{mapsto}\ 1\ x$) is matched the singleton heap that has value x in location 1. The quantified pattern $\exists x \,.\, 1 \mapsto x$ is matched by any singleton heap that has any value in location 1. In other words, the quantifier $\exists x$ *abstracts away* the value x.

Example 3.8. Patterns can also represent predicates/assertions. In matching μ-logic, we use $\bot$ and $\top$ to represent the truth values, where $\bot$ is a primitive construct in Definition 3.1 and $\top$ can be defined as a notation (see Definition 3.15). Intuitively, $\bot$ is matched by no elements while $\top$ is matched by all elements. A pattern is called a *predicate pattern* if it is either $\bot$ or $\top$.

Example 3.9. Let $\Sigma = \{\mathsf{emp}, \mathsf{mapsto}, \mathsf{merge}, \mathsf{gt}\}$, where gt represents the greater-than predicate. For notational simplicity, we write $\mathsf{gt}\ x\ y$ as $x > y$. Let $x > y$ be $\top$ if x is greater than y, and $\bot$ otherwise. Then, $\exists x \,.\, 1 \mapsto x \wedge x > 2$ is matched by any singleton heap that has any value greater than 2 in location 1. Indeed, if x is greater than 2, then $1 \mapsto x \wedge x > 2 = 1 \mapsto x \wedge \top = 1 \mapsto x$. Otherwise, $1 \mapsto x \wedge x > 2 = 1 \mapsto x \wedge \bot = \bot$.

In literature, patterns such as $1 \mapsto x \wedge x > 2$ are called *constrained terms*, where $1 \mapsto x$ is a term that describes the data structure and $x > 2$ is a logical constraint that restricts the variables in the term. In matching μ-logic, both terms and logical constraints are unified by patterns. Constrained terms *are* matching μ-logic patterns *as is*.

Finally, we discuss fixpoints in matching μ-logic. The least fixpoint pattern $\mu X \,.\, \varphi$ represents the set X that is the smallest solution (w.r.t. set inclusion) of the equation $X = \varphi$. Note that X may occur recursively in φ. The existence of the least fixpoints is guaranteed by requiring φ be positive in X.

Example 3.10. The least fixpoint $\mu X \,.\, X$ represents the set X that is the smallest solution of $X = X$. Since any set is a solution of the equation, the smallest solution is the empty set. Therefore, $\mu X \,.\, X$ is the same as $\bot$.

Example 3.11. The least fixpoint $\mu X \,.\, \mathsf{zero} \vee (\mathsf{succ}\ X)$ represents the smallest solution X of $X = \mathsf{zero} \vee (\mathsf{succ}\ X)$. In other words, X is the smallest set that includes zero and is closed under succ. Therefore, $\mu X \,.\, \mathsf{zero} \vee (\mathsf{succ}\ X)$ yields exactly the set of all natural numbers: $\{\mathsf{zero}, (\mathsf{succ}\ \mathsf{zero}), (\mathsf{succ}\ (\mathsf{succ}\ \mathsf{zero})), \dots\}$.

Example 3.12. Let us consider $\mu X \,.\, \neg X$, where $\neg X$ is the negation of X, defined as $X \to \bot$ (see Definition 3.15). Semantically, $\neg X$ represents the complement of X. Then, $\mu X \,.\, \neg X$ represents the smallest solution of $X = \neg X$. However, no set equals to its complement. Thus, $X = \neg X$ has no solutions and $\mu X \,.\, \neg X$ is meaningless. Indeed, $\mu X \,.\, \neg X$ is *not* a wellformed pattern because $\neg X$ is not positive in X.

From the above examples, we can see that matching μ-logic provides a uniform syntax for patterns that can be matched by zero, one, or more elements. Patterns can be built to match data structures that have certain forms or meet certain logical constraints. By introducing proper notations, many familiar syntaxes become wellformed matching μ-logic patterns *as is*, including FOL terms, predicates/assertions, constrained terms, and the heap formulas in separation logic.

In the rest of the section, we formally define free variables, capture-avoiding substitution, and some common notations that can be derived from the primitive pattern syntax.

Definition 3.13 (Free Variables)**.** The set of *free variables* in a pattern φ is denoted by $\mathbb{FV}(\varphi) \subseteq \mathbb{EV} \cup \mathbb{SV}$ and is inductively defined as follows:

- $\mathbb{FV}(x) = \{x\}$ for $x \in \mathbb{EV}$;
- $\mathbb{FV}(X) = \{X\}$ for $X \in \mathbb{SV}$;
- $\mathbb{FV}(\sigma) = \{\sigma\}$ for $\sigma \in \Sigma$;
- $\mathbb{FV}(\varphi_1\ \varphi_2) = \mathbb{FV}(\varphi_1) \cup \mathbb{FV}(\varphi_2)$;
- $\mathbb{FV}(\bot) = \emptyset$;
- $\mathbb{FV}(\varphi_1 \to \varphi_2) = \mathbb{FV}(\varphi_1) \cup \mathbb{FV}(\varphi_2)$;
- $\mathbb{FV}(\exists x\,.\,\varphi) = \mathbb{FV}(\varphi) \setminus \{x\}$;
- $\mathbb{FV}(\mu X\,.\,\varphi) = \mathbb{FV}(\varphi) \setminus \{X\}$.

If $\mathbb{FV}(\varphi) = \emptyset$, we call φ a *closed pattern.*

Definition 3.13 is given in the usual way. Note that $\exists x$ and μX are the only two binders in the (primitive) syntax of patterns. Following convention, we allow α*-renaming* and regard α*-equivalent* patterns as identical. For example, $\exists x\,.\,1 \mapsto x$ and $\exists y\,.\,1 \mapsto y$ are the same pattern.

We define *capture-avoiding substitution*, where α-renaming can happen implicitly to prevent variable capture. Formally,

Definition 3.14 (Capture-Avoiding Substitution)**.** We use $\varphi[\psi/x]$ to denote the result of substituting ψ for x in φ, where α-renaming happens implicitly to avoid variable capture, in the following usual way:

- $x[\psi/x] = \psi$;
- $y[\psi/x] = y$ if $y \in \mathbb{EV}$ is distinct from x;
- $Y[\psi/x] = Y$ for $Y \in \mathbb{SV}$;
- $\sigma[\psi/x] = \sigma$ for $\sigma \in \Sigma$;
- $(\varphi_1\ \varphi_2)[\psi/x] = (\varphi_1[\psi/x])\ (\varphi_2[\psi/x])$;
- $\bot[\psi/x] = \bot$;
- $(\varphi_1 \to \varphi_2)[\psi/x] = \varphi_1[\psi/x] \to \varphi_2[\psi/x]$;
- $(\exists x\,.\,\varphi)[\psi/x] = \exists x\,.\,\varphi$, called *variable shadowing*;
- $(\exists y\,.\,\varphi)[\psi/x] = \exists z\,.\,\varphi[z/y][\psi/x]$, where $z \notin \mathbb{FV}(\varphi) \cup \mathbb{FV}(\psi)$;

- $(\mu Y \,.\, \varphi)[\psi/x] = \mu Z \,.\, \varphi[Z/Y][\psi/x]$, where $Z \notin \mathbb{FV}(\varphi) \cup \mathbb{FV}(\psi)$.

Note that in the last two cases, we first rename $\exists y$ and μY to $\exists z$ and μZ, respectively, for variables z and Z that do not occur freely in ψ or φ. Similarly, we use $\varphi[\psi/X]$ to denote the result of substituting ψ for a set variable X in φ; the formal definitions are as follows:

- $x[\psi/X] = x$ for $x \in \mathbb{EV}$;
- $X[\psi/X] = \psi$;
- $Y[\psi/X] = Y$ if $Y \in \mathbb{SV}$ is distinct from X;
- $\sigma[\psi/X] = \sigma$ for $\sigma \in \Sigma$;
- $(\varphi_1\ \varphi_2)[\psi/X] = (\varphi_1[\psi/X])\ (\varphi_2[\psi/X])$;
- $\bot[\psi/X] = \bot$;
- $(\varphi_1 \to \varphi_2)[\psi/X] = \varphi_1[\psi/X] \to \varphi_2[\psi/X]$;
- $(\exists x \,.\, \varphi)[\psi/X] = \exists z \,.\, \varphi[z/x][\psi/X]$, where $z \notin \mathbb{FV}(\varphi) \cup \mathbb{FV}(\psi)$;
- $(\mu X \,.\, \varphi)[\psi/X] = \mu X \,.\, \varphi$, called *variable shadowing*;
- $(\mu Y \,.\, \varphi)[\psi/X] = \mu Z \,.\, \varphi[Z/Y][\psi/X]$, where $Z \notin \mathbb{FV}(\varphi) \cup \mathbb{FV}(\psi)$.

Finally, we define the following common connectives as *notations*.

Definition 3.15. Let us define the following notations:

$\neg\varphi \equiv \varphi \to \bot$	Negation
$\top \equiv \neg\bot$	Top (Logical Truth)
$\varphi_1 \vee \varphi_2 \equiv \neg\varphi_1 \to \varphi_2$	Disjunction
$\varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$	Conjunction
$\varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$	"If and Only If"
$\forall x \,.\, \varphi \equiv \neg\exists x \,.\, \neg\varphi$	Universal Quantification
$\nu X \,.\, \varphi \equiv \neg\mu X \,.\, \neg\varphi[\neg X/X]$	Greatest Fixpoints

3.2 Matching μ-Logic Semantics

The semantics of matching μ-logic is based on *pattern matching*. The intuition is that a pattern represents the set of elements that *match* it.

Before we define the semantics of matching μ-logic, we need to formalize the notion of matching μ-logic models.

3.2.1 Matching μ-Logic Models

Definition 3.16 (Matching μ-Logic Models)**.** Given a signature Σ, a Σ-*model* or simply a *model* is a tuple $(M, @_M, \{\sigma_M\}_{\sigma \in \Sigma})$ that consists of the following three components:

1. a nonempty *carrier set* M, whose elements are denoted a, b, $\ldots$;
2. a binary *application function* $@_M \colon M \times M \to \mathcal{P}(M)$, where $\mathcal{P}(M)$ is the powerset of M;
3. a *symbol interpretation* $\sigma_M \subseteq M$ as a subset, for each $\sigma \in \Sigma$.

For notational simplicity, we use the same letter M to refer to the model defined above.

Example 3.17. The simplest example of a model is a one for the empty signature; that is, $\Sigma = \emptyset$. In this case, a model M consists of:

- a nonempty carrier set which we also denote by M; and
- an application function $@_M \colon M \times M \to \mathcal{P}(M)$.

Since the signature is empty, M does not have interpretations of symbols.

Functional Interpretation versus Powerset Interpretation

In matching μ-logic, the application function $@_M$ returns subsets of M. It is therefore different from FOL, where a binary functional symbol is interpreted as a binary function that returns elements in M, and not subsets of M. That is, if @ was a binary functional symbol in FOL, its interpretation would be $@_M \colon M \times M \to M$, and not $@_M \colon M \times M \to \mathcal{P}(M)$ as in Definition 3.16. We use *functional interpretation* to refer to the way how FOL models interpret their functional symbols, and *powerset interpretation* how matching μ-logic models interpret their symbols.

Powerset interpretation generalizes functional interpretation. Indeed, due to the one-to-one correspondence between an element a and the singleton set $\{a\}$, we can treat functional interpretation as a special instance of powerset interpretation as follows:

	Functional Interpretation	**Powerset Interpretation**
Carrier set	M	$\{\{a\} \mid a \in M\} \subseteq \mathcal{P}(M)$
An element	$a \in M$	$\{a\} \in \mathcal{P}(M)$
A function	$f \colon M \times \cdots \times M \to M$	$\bar{f} \colon \mathcal{P}(M) \times \cdots \times \mathcal{P}(M) \to \mathcal{P}(M)$
Function application	$f(a_1, \ldots, a_n)$	$\bar{f}(\{a_1\}, \ldots, \{a_n\})$

We define the *pointwise extension* of the application function in a matching μ-logic model.

Definition 3.18. Let M be a matching μ-logic model and $@_M \colon M \times M \to \mathcal{P}(M)$ be its application function. Its *pointwise extension* $\overline{@_M} \colon \mathcal{P}(M) \times \mathcal{P}(M) \to \mathcal{P}(M)$ is defined as follows:

$$\overline{@_M}(A, B) = \bigcup_{a \in A} \bigcup_{b \in B} a \, @_M \, b \quad \text{for } A, B \subseteq M$$

Proposition 3.19. *$\overline{@_M}(A, \emptyset) = \overline{@_M}(\emptyset, B) = \emptyset$, for any $A, B \subseteq M$. Here, $\emptyset$ denotes the empty set.*

Proposition 3.20. *$\overline{@_M}(\{a\}, \{b\}) = @_M(a, b)$, for any $a, b \in M$.*

Proposition 3.21. *For any $A, B, C \subseteq M$, we have*

- $\overline{@_M}(A \cup B, C) = \overline{@_M}(A, C) \cup \overline{@_M}(B, C)$*, and*
- $\overline{@_M}(A, B \cup C) = \overline{@_M}(A, C) \cup \overline{@_M}(A, C)$*.*

We use the example of *applicative structures* to elaborate on the relation between functional and powerset interpretations.

Definition 3.22 (Applicative Structures)**.** An *applicative structure* is a pair $(A, @_A)$ that consists of:

- a nonempty carrier set A; and
- a binary application function $@_A \colon A \times A \to A$.

We use the same letter A to denote the above applicative structure.

The only difference between applicative structures and matching μ-logic models of the empty signature (Example 3.17) is that the range of $@_A$ in an applicative structure A is A itself, while the range of $@_M$ in a matching μ-logic model M is $\mathcal{P}(M)$. The following proposition shows that applicative structures are a special instance of matching μ-logic models.

Proposition 3.23. *Let $(A, @_A)$ be an applicative structure, from which we can derive a matching μ-logic model of the empty signature, where*

- *the carrier set $M^A = A$; and*
- *the application function $@_{M^A} \colon A \times A \to \mathcal{P}(A)$ is defined by $a \mathbin{@_{M^A}} b = \{a \mathbin{@_A} b\}$ for all $a, b \in M$.*

Recall that $\overline{@_{M^A}} \colon \mathcal{P}(A) \times \mathcal{P}(A) \to \mathcal{P}(A)$ is the pointwise extension of $@_{M^A}$ (see Definition 3.18). Then, $(A, @_A)$ is isomorphic to the derived matching μ-logic model in the following sense:

	Applicative Structure A	***Matching*** μ***-Logic Model*** M^A
Carrier set	A	$M^A = A$*, whose powerset is* $\mathcal{P}(A)$
An element	$a \in A$	$a \in A$*, or alternatively,* $\{a\} \in \mathcal{P}(A)$
Application function	$@_A \colon A \times A \to A$	$\overline{@_{M^A}} \colon \mathcal{P}(A) \times \mathcal{P}(A) \to \mathcal{P}(A)$
Function application	$a \mathbin{@_A} b$	$\{a\} \mathbin{\overline{@_{M^A}}} \{b\}$

To conclude, functional interpretation is a special instance of powerset interpretation under the *element-singleton correspondence* shown as above.

3.2.2 Variable Valuations and Pattern Interpretations

Given a model M, a *variable valuation* is a function that maps element variables to elements of M and set variables to subsets of M, respectively. We define variable valuations formally as follows:

Definition 3.24 (Matching μ-Logic Valuations)**.** Let M be a model as in Definition 3.16. An M*-valuation* or simply a *valuation* is a function $\rho \colon \mathbb{EV} \cup \mathbb{SV} \to M \cup \mathcal{P}(M)$ that maps every element variable $x \in \mathbb{EV}$ to an element $\rho(x) \in M$ and every set variable $X \in \mathbb{SV}$ to a subset $\rho(X) \subseteq M$.

Definition 3.25 (Pattern Interpretation)**.** Let M and ρ be a model and a valuation as in Definition 3.24. We inductively define *interpretation of patterns*, written $|\varphi|_{M,\rho} \subseteq M$, as follows:

1. $|x|_{M,\rho} = \{\rho(x)\}$ for $x \in \mathbb{EV}$;
2. $|X|_{M,\rho} = \rho(X)$ for $X \in \mathbb{SV}$;
3. $|\sigma|_{M,\rho} = \sigma_M$ for $\sigma \in \Sigma$;
4. $|\varphi_1\, \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \,\overline{@_M}\, |\varphi_2|_{M,\rho}$, where $\overline{@_M}$ is the pointwise extension in Definition 3.18;
5. $|\bot|_{M,\rho} = \emptyset$;
6. $|\varphi_1 \to \varphi_2|_{M,\rho} = M \setminus (|\varphi_1|_{M,\rho} \setminus |\varphi_2|_{M,\rho})$;
7. $|\exists x \,.\, \varphi|_{M,\rho} = \bigcup_{a \in M} |\varphi|_{M,\rho[a/x]}$;
8. $|\mu X \,.\, \varphi|_{M,\rho} = \mathbf{lfp}(A \mapsto |\varphi|_{M,\rho[A/X]})$;

where $\rho[a/x]$ (resp. $\rho[A/X]$) is the valuation ρ' with $\rho'(x) = a$ (resp. $\rho'(X) = A$) and agreeing with ρ on all the other variables. We use $\mathbf{lfp}(A \mapsto |\varphi|_{M,\rho[A/X]})$ to denote the least solution of the equation $A = |\varphi|_{M,\rho[A/X]}$, w.r.t. set inclusion.

Proposition 3.26. *In Definition 3.25, $|\mu X \,.\, \varphi|_{M,\rho}$ is well-defined.*

Proof. **To do** (??) □

It is expected that the interpretation of a pattern φ does not depend on the valuations of the variables that do not occur free in φ. Formally

Proposition 3.27. *$|\varphi|_{M,\rho} = |\varphi|_{M,\rho'}$ if*

1. *$\rho(x) = \rho'(x)$ for all $x \in \mathbb{EV} \cap \mathbb{FV}(\varphi)$; and*
2. *$\rho(X) = \rho'(X)$ for all $X \in \mathbb{SV} \cap \mathbb{FV}(\varphi)$.*

For notational simplicity, we write $\rho =_{\mathbb{FV}(\varphi)} \rho'$ if both (1) and (2) hold.

For notational simplicity, we abbreviate $|\varphi|_{M,\rho}$ as $|\varphi|_\rho$ when M is clear from the context. We further abbreviate it as $|\varphi|$ if φ is closed, in which case ρ is irrelevant (Proposition 3.27).

In the following, we list the semantics of matching logic notations that are introduced in Section 3.1.

Proposition 3.28. *Let M be a model and ρ be a valuation. The following hold:*

1. $|\neg\varphi|_{M,\rho} = M \setminus |\varphi|_{M,\rho}$;
2. $|\top|_{M,\rho} = M$;
3. $|\varphi_1 \vee \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \cup |\varphi_2|_{M,\rho}$;
4. $|\varphi_1 \wedge \varphi_2|_{M,\rho} = |\varphi_1|_{M,\rho} \cap |\varphi_2|_{M,\rho}$;
5. $|\varphi_1 \leftrightarrow \varphi_2|_{M,\rho} = M \setminus (|\varphi_1|_{M,\rho} \triangle |\varphi_2|_{M,\rho})$;
6. $|\forall x \,.\, \varphi|_{M,\rho} = \bigcap_{a \in M} |\varphi|_{M,\rho[a/x]}$;
7. $|\nu X \,.\, \varphi|_{M,\rho} = \mathbf{gfp}(A \mapsto |\varphi|_{M,\rho[A/X]})$;

where "$\triangle$" is the set symmetric difference operation (see Section 2.1) and **gfp** *is the greatest fixpoint operation.*

To do Add some examples here. (**??**)

Example 3.29. $|\mu X \,.\, X|_M = \emptyset$.

Proof. **To do** (**??**) □

Example 3.30. $|\nu X \,.\, X|_M = M$.

Proof. **To do** (**??**) □

Example 3.31. Let zero and succ be two symbols. Let M be a model whose carrier set includes $\mathbb{N}$, the set of natural numbers. Suppose M satisfy the following conditions:

- $|\mathsf{zero}|_M = \{0\}$;
- $|\mathsf{succ}\ x|_{M,\rho} = \{\rho(x) + 1\}$, for $\rho(x) \in \mathbb{N}$.

Then, $|\mu N \,.\, \mathsf{zero} \vee \mathsf{succ}\ N|_M = \mathbb{N}$.

Proof. **To do** (**??**) □

Functional Patterns

Given M and ρ, we say that a pattern φ is a *functional pattern* with respect to M and ρ, if $|\varphi|_{M,\rho}$ is a singleton set, i.e., it includes exactly one element. We say that φ is a functional pattern with respect to M, if it is a functional pattern with respect to any valuations. By convention, we use the metavariable t and its variants (t', t'', t_1, t_2, etc.) to denote functional patterns. Functional patterns are the matching logic counterpart of *terms* in FOL.

Predicate Patterns

Given M and ρ, we say that a pattern φ is a *predicate pattern* with respect to M and ρ, if $|\varphi|_{M,\rho} \in \{M, \emptyset\}$. If $|\varphi|_{M,\rho}$, we say that φ *holds*. Otherwise, we say that φ *does not hold* or *fails*. Predicate patterns are the matching logic counterpart of *formulas* in FOL.

3.2.3 Validity and Satisfiability

In this section, we define the relations of *validity* and *satisfiability*. Both relations have to do with the semantics of matching logic, and not its proof system (which is discussed in Section 3.3). As we will shortly see, validity and satisfiability are dual to each other.

Definition 3.32 (Matching μ-Logic Validity)**.** Let M be a model and φ be a pattern. We say that φ is *M-valid*, written $M \vDash \varphi$, if $|\varphi|_{M,\rho} = M$ for all ρ. We say that φ is *valid*, written $\vDash \varphi$, if $M \vDash \varphi$ for all M.

In other words, φ is M-valid if its interpretation in M is the entire carrier set. It is worth to note that φ is not M-valid *does not imply* that there exists a valuation ρ such that $|\varphi|_{M,\rho} = \emptyset$. Here is a counterexample.

Example 3.33. Let x be an element variable and M be a model whose carrier set is $\{a, b\}$. Then the following propositions are all true:

1. x is not M-valid;
2. $\neg x$ is not M-valid;
3. $|x|_{M,\rho} \neq M$ for all ρ;
4. $|x|_{M,\rho} \neq \emptyset$ for all ρ.

Let Γ be a set of patterns. We write $M \vDash \Gamma$ to mean that $M \vDash \varphi$ for all $\varphi \in \Gamma$. We often call Γ a *theory*, and patterns in Γ *axioms*.

Axioms are used to restrict the models by enforcing them to satisfy the axiomatized properties. In the following, we give some examples about using axioms to enforce the models to have certain properties.

Example 3.34. Let M be any model. Then, M has exactly one element if and only if $M \vDash \{x\}$, where x is an element variable.

Proof. For the "if" part, note that for any valuation ρ, $|x|_{M,\rho} = \{\rho(x)\}$. Since $M \vDash \{x\}$, we have $\{\rho(x)\} = M$ for any valuation ρ. Therefore, M must have exactly one element in its carrier set.

For the "only if" part, assume that the carrier set $M = \{\star\}$. Then for any valuation ρ, $\rho(x) = \star$, and thus $|x|_{M,\rho} = \{\rho(x)\} = \{\star\} = M$. Therefore, $M \vDash \{x\}$. $\square$

Example 3.35. Let M be any model. Then, M has at least two elements in its carrier set if and only if $M \vDash \{\exists x \,.\, \neg x\}$.

Proof. For the "if" part, let us assume that M has exactly one element (note that the carrier set cannot be empty) and we look for a contradiction. Suppose $M = \{\star\}$. Then for any valuation ρ, $|\exists x \,.\, \neg x|_{M,\rho} = \bigcup_{a \in M} |\neg x|_{M,\rho[a/x]} = |\neg x|_{M,\rho[\star/x]} = M \setminus \{\star\} = \emptyset$, which contradicts the fact that $M \vDash \{\exists x \,.\, \neg x\}$. This contradiction proves that M must have at least two elements in its carrier set.

For the "only if" part, let $\star_1$ and $\star_2$ be two different elements in the carrier set of M. Then we have

$$|\neg x|_{M,\rho[\star_1/x]} \cup |\neg x|_{M,\rho[\star_2/x]} = (M \setminus \{\star_1\}) \cup (M \setminus \{\star_2\}) = M$$

Therefore, $M \supseteq |\exists x \,.\, \neg x|_{M,\rho} = \bigcup_{a \in M} |\varphi|_{M,\rho[a/x]} \supseteq |\neg x|_{M,\rho[\star_1/x]} \cup |\neg x|_{M,\rho[\star_2/x]} = M$. It follows that $|\exists x \,.\, \neg x|_{M,\rho} = M$. $\square$

Definition 3.36 (Matching μ-Logic Validity Cont.)**.** Let Γ be a theory and ψ be a pattern. We write $\Gamma \vDash \psi$ to mean that $M \vDash \Gamma$ implies $M \vDash \psi$, for all models M.

In literature, $\Gamma \vDash \psi$ is also known as *semantic entailment*.

Proposition 3.37. *$\emptyset \vDash \psi$ if and only if $\vDash \psi$.*

Proof. Simply note that any model $M \vDash \emptyset$. $\square$

Satisfiability is the dual concept of validity. While validity states that a pattern is interpreted as the entire carrier set, satisfiability states that it is interpreted as a *nonempty* set. Formally,

Definition 3.38 (Matching μ-Logic Satisfiability)**.** Let M be a model and φ be a pattern. We say that φ is *satisfiable* in M if there exists a valuation ρ such that $|\varphi|_{M,\rho} \neq \emptyset$. We call such ρ a *witness valuation* and any element $a \in |\varphi|_{M,\rho}$ a *witness element*. We say that φ is *satisfiable* if there exists a model M such that φ is satisfiable in M. We call such M a *witness model* of φ.

Unsatisfiability is naturally defined as the negation of satisfiability. For clarity, we state its formal definition below:

Definition 3.39. Let M be a model and φ be a pattern. We say that φ is *unsatisfiable* in M, if $|\varphi|_{M,\rho} = \emptyset$ for all ρ. We say that φ is *unsatisfiable*, if it is unsatisfiable in all M.

Definition 3.40 (Matching μ-Logic Satisfiability, Cont.)**.** Let Γ be a theory and φ be a pattern. We say that φ is Γ-*satisfiable* if it is satisfiable in M for some $M \vDash \Gamma$. We say that φ is Γ-*unsatisfiable* if it is unsatisfiable in M for any $M \vDash \Gamma$.

The following theorem establishes the duality between validity and satisfiability.

Theorem 3.41 (Duality between Validity and Satisfiability)**.** *For any M and φ, φ is M-valid if and only if $\neg\varphi$ is unsatisfiable in M.*

Proof. We have the following reasoning:

$$\begin{array}{rcll}
\varphi \text{ is } M\text{-valid} & \iff & M \vDash \varphi & \\
& \iff & |\varphi|_{M,\rho} = M \text{ for any valuation } \rho & \\
& \iff & |\neg\varphi|_{M,\rho} = \emptyset \text{ for any valuations } \rho & \text{Proposition 3.28(1)} \\
& \iff & \neg\varphi \text{ is unsatisfiable in } M &
\end{array}$$

□

The following corollaries are straightforward and used more often.

Corollary 3.42. *For any Γ and φ, $\Gamma \vDash \varphi$ if and only if $\neg\varphi$ is Γ-unsatisfiable.*

Proof. We have the following reasoning:

$$\begin{array}{rcll}
\Gamma \vDash \varphi & \iff & M \vDash \varphi \text{ for any } M \vDash \Gamma & \\
& \iff & \neg\varphi \text{ is unsatisfiable in } M\text{, for any } M \vDash \Gamma & \text{Theorem 3.41} \\
& \iff & \neg\varphi \text{ is } \Gamma\text{-unsatisfiable} &
\end{array}$$

□

Corollary 3.43. *For any φ, φ is valid if and only if $\neg\varphi$ is unsatisfiable.*

Proof. Let $\Gamma = \emptyset$ in Corollary 3.42. □

In the following, we define two important decision problems, one about validity and the other about satisfiability.

Definition 3.44. The decision problem VALID? asks whether $\Gamma \vDash \varphi$ for given Γ and φ.

Definition 3.45. The decision problem SAT? asks whether φ is satisfiable in some M such that $M \vDash \Gamma$, for given Γ and φ.

Theorem 3.46. *Both decision problems* VALID*? and* SAT*? are undecidable.*

Proof. **To do** (??) □

Theorem 3.46 states that it is in general undecidable to check validity or satisfiability in matching μ-logic.

Definition 3.47. Let $\mathcal{T}$ be a set of theories and $\mathcal{P}$ be a set of patterns. The decision problem $\mathsf{VALID?}(\mathcal{T},\mathcal{P})$ asks whether $\Gamma \vDash \varphi$ for given $\Gamma \in \mathcal{T}$ and $\varphi \in \mathcal{P}$.

Definition 3.48. Let $\mathcal{T}$ be a set of theories and $\mathcal{P}$ be a set of patterns. The decision problem $\mathsf{SAT?}(\mathcal{T},\mathcal{P})$ asks whether φ is satisfiable in some M such that $M \vDash \Gamma$, for given $\Gamma \in \mathcal{T}$ and $\varphi \in \mathcal{P}$.

Definition 3.49. A pattern set $\mathcal{P}$ is *negation-closed*, if $\neg\varphi \in \mathcal{P}$ for any $\varphi \in \mathcal{P}$.

Theorem 3.50. *For any $\mathcal{T}$ and $\mathcal{P}$ that is negation-closed,* $\mathsf{VALID?}(\mathcal{T},\mathcal{P})$ *is decidable if and only if* $\mathsf{SAT?}(\mathcal{T},\mathcal{P})$ *is decidable.*

Proof. Let $\Gamma \in \mathcal{T}$ and $\varphi \in \mathcal{P}$ be any theory and pattern, respectively.

We first prove the "if" part. Suppose A is an algorithm that checks whether φ is Γ-satisfiable and returns either ✓ or ✗, for inputs $\varphi \in \mathcal{P}$ and $\Gamma \in \mathcal{T}$. Construct a new algorithm A' as follows. For any inputs $\varphi \in \mathcal{P}$ and $\Gamma \in \mathcal{T}$, A' runs the algorithm A on the inputs $\neg\varphi$ and Γ. If A returns ✓, A' returns ✗. If A returns ✗, A' returns ✓. Since $\mathcal{P}$ is negation-closed, $\neg\varphi$ is also a member of $\mathcal{P}$, and thus A' is a well-defined algorithm that terminates on all inputs.

We claim that A' is a decision procedure of $\mathsf{VALID?}(\mathcal{T},\mathcal{P})$. Indeed, for inputs Γ and φ, if A' returns ✓, then A returns ✗ for inputs Γ and $\neg\varphi$, which implies that $\neg\varphi$ is Γ-unsatisfiable. By Corollary 3.42, $\Gamma \vDash \varphi$. On the other hand, if A' returns ✗, then A returns ✓ for inputs Γ and $\neg\varphi$, which implies that $\neg\varphi$ is Γ-satisfiable. Also by Corollary 3.42, $\Gamma \nvDash \varphi$. Therefore, $\mathsf{VALID?}(\mathcal{T},\mathcal{P})$ is decidable by algorithm A'.

We next prove the "only if" part following the same idea. Suppose B is an algorithm that checks whether $\Gamma \vDash \varphi$ and returns either ✓ or ✗, for inputs $\Gamma \in \mathcal{T}$ and $\varphi \in \mathcal{P}$. Construct a new algorithm B' that calls B on $\neg\varphi$ and Γ and flips the return value of B. Then by Corollary 3.42, B checks whether φ is Γ-satisfiable. Therefore, $\mathsf{SAT?}(\mathcal{T},\mathcal{P})$ is decidable.

To conclude, $\mathsf{VALID?}(\mathcal{T},\mathcal{P})$ and $\mathsf{SAT?}(\mathcal{T},\mathcal{P})$ have the same decidability for any pattern set $\mathcal{P}$ that is negation-closed. $\square$

3.3 Matching μ-Logic Proof System and the Soundness Theorem

In this section, we present the matching μ-logic proof system and prove its soundness theorem.

3.3.1 Matching μ-Logic Proof System

The matching μ-logic proof system presented in Figure 3.1 is a Hilbert-style proof system that consists of 13 proof rules. The proof system defines a provability relation between a theory Γ and a pattern φ, denoted by $\Gamma \vdash \varphi$. We assume the reader is familiar with the basic concepts of Hilbert-style proof systems and the definition of Hilbert-style proofs. If not, we refer to **todo**.

To understand the proof system, we first need to define *contexts* and *application contexts*.

Definition 3.51 (Contexts)**.** A *context* C is a pattern with a distinguished set variable $\square$, called the *placeholder variable*, where $\square$ has exactly one free occurrence and no bound occurrence in C. We write $C[\varphi]$ to mean the result of replacing $\square$ with φ.

Example 3.52. Let $\varphi \equiv x \wedge y$ in the following statements.

1. Let $C_1 \equiv \square$. Then $C_1[\varphi] \equiv x \wedge y$.

(Propositional 1)	$\varphi_1 \to (\varphi_2 \to \varphi_1)$
(Propositional 2)	$\varphi_1 \to (\varphi_2 \to \varphi_3) \to (\varphi_1 \to \varphi_2) \to (\varphi_1 \to \varphi_3)$
(Propositional 3)	$((\varphi \to \bot) \to \bot) \to \varphi$
(Modus Ponens)	$\dfrac{\varphi_1 \quad \varphi_1 \to \varphi_2}{\varphi_2}$
(∃-Quantifier)	$\varphi[y/x] \to \exists x \,.\, \varphi$
(∃-Generalization)	$\dfrac{\varphi_1 \to \varphi_2}{(\exists x.\varphi_1) \to \varphi_2}$ if $x \notin \mathbb{FV}(\varphi_2)$
(Propagation$_\bot$)	$C[\bot] \to \bot$
(Propagation$_\vee$)	$C[\varphi_1 \vee \varphi_2] \to C[\varphi_1] \vee C[\varphi_2]$
(Propagation$_\exists$)	$C[\exists x.\varphi] \to \exists x.C[\varphi]$ if $x \notin \mathbb{FV}(C[\exists x.\varphi])$
(Framing)	$\dfrac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}$
(Substitution)	$\dfrac{\varphi}{\varphi[\psi/X]}$
(Pre-Fixpoint)	$\varphi[\mu X \,.\, \varphi/X] \to \mu X \,.\, \varphi$
(Knaster-Tarski)	$\dfrac{\varphi[\psi/X] \to \psi}{\mu X \,.\, \varphi \to \psi}$
(Existence)	$\exists x.\, x$
(Singleton)	$\neg(C_1[x \wedge \varphi] \wedge C_2[x \wedge \neg\varphi])$

In the above proof rules, C, C_1, and C_2 are application contexts (see Definition 3.53).

Figure 3.1: Matching μ-Logic Proof System

2. Let $C_2 \equiv \square \wedge z$. Then $C_2[\varphi] \equiv (x \wedge y) \wedge z$.

3. Let $C_3 \equiv y\, \square$. Then $C_3[\varphi] \equiv y\,(x \wedge y)$.

4. Let $C_4 \equiv \exists z \,.\, \square$. Then $C_4[\varphi] \equiv \exists z \,.\, x \wedge y$.

5. Let $C_5 \equiv \exists x \,.\, \square$. Then $C_5[\varphi] \equiv \exists x \,.\, x \wedge y$.

Note that $C[\varphi]$ is not the same as (capture-avoiding) substitution $C[\varphi/\square]$, although in many cases the two are identical. In Example 3.52, the first four cases still hold for substitution, but the fifth case shows the difference, where $C_5[\varphi/\square] \equiv \exists z \,.\, x \wedge y$. The quantifier $\exists x$ is renamed to $\exists z$ for a fresh name z to avoid variable capture. However, for $C_5[\varphi]$, such renaming is not happening. In literature, people often refer to the operations such as $C_5[\varphi]$ as *replacement*, making it different from (capture-avoiding) substitution.

Replacement is used less often than substitution. Unlike substitution whose usage is often necessary and cannot be avoided, replacement is mostly voluntarily used to simply the *presentation* of certain concepts and definitions, and can be avoided, at the cost of using potentially more complex presentations and notations. For example, the matching μ-logic proof system in Figure 3.1 uses both substitution (such as ($\exists$-QUANTIFIER)) and replacement (such as (FRAMING)), but the latter can be avoided if we split (FRAMING) into several sub-rules, as shown in **??**. Therefore, replacement should be understood as a mechanism *at the metalevel* that helps to simply our presentation of matching μ-logic.

Among all contexts, there is one type of contexts in which we have special interest. They are called *application contexts* and we formally define them below.

Definition 3.53 (Application Contexts)**.** An *application context* C is inductively defined by the following two rules:

1. C is an application context if it is $\square$, i.e., the identity context;

2. C is an application context if it is an application pattern of the form $C'\,\varphi$ or $\varphi\, C'$, where C' is an application context and φ is any pattern that has no occurrence of the placeholder variable $\square$.

Intuitively speaking, an application context is one where the path from the root to $\square$ includes only applications. No other constructs are allowed on the path. In Example 3.52, only C_3 is an application context. All the others are contexts but not application contexts.

Now, we are ready to discuss the proof rules in the matching μ-logic proof system in Figure 3.1. We divide the proof rules in the following four categories, based on the types of formal reasoning that they support:

1. FOL reasoning;

2. frame reasoning;

3. fixpoint reasoning;

4. miscellaneous proof rules.

We briefly explain each proof rule category in the following. More detailed discussion on how to apply these proof rules is given in **??**.

Proof Rules that Support FOL Reasoning

The first six proof rules in Figure 3.1 form a complete proof system for predicate logic **todo**. In particular, the first four proof rules, i.e., from (Propositional 1) to (Modus Ponens), are credited to Alonzo Church, who formulated a complete Hilbert-style proof system for propositional reasoning that is based on implication $\to$ and falsum $\bot$. The following two proof rules, ($\exists$-Quantifier) and ($\exists$-Generalization), reason about the existential quantifiers.

Example 3.54. The following is the "hello world" example of a Hilbert-style proof. We use the proof rules in Figure 3.1 to prove that $\vdash \varphi \to \varphi$.

1	$\varphi \to (\varphi \to \varphi)$	(Propositional 1)
2	$\varphi \to ((\varphi \to \varphi) \to \varphi)$	(Propositional 1)
3	$(\varphi \to ((\varphi \to \varphi) \to \varphi) \to (\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi)$	(Propositional 2)
4	$(\varphi \to (\varphi \to \varphi)) \to (\varphi \to \varphi)$	(Modus Ponens) on 2,3
5	$\varphi \to \varphi$	(Modus Ponens) on 1,4

To do Add some discussion on proof rule schemas. (**??**)

The proof rule ($\exists$-Quantifier) is also known as the *instantiation rule*, where the left-hand side $\varphi[y/x]$ is an instance of the right-hand side $\exists x \,.\, \varphi$, by letting the binding variable x to be y. However, it is worth noting that the following attempt to generalize the ($\exists$-Quantifier) rule will *fail*:

$$(\exists\text{-Quantifier'}) \qquad \varphi[\psi/x] \to \exists x \,.\, \varphi$$

Proposition 3.55. *($\exists$-Quantifier') has instances that are not valid.*

Proof. Consider the following instance of ($\exists$-Quantifier'):

$$\neg\bot \to \exists x \,.\, \neg x$$

where we let φ be $\neg x$ and ψ be $\bot$. We show that the above instance is not valid by constructing a model M whose carrier set has only one element, say $\star$. Obviously, any M-valuation maps element variables to the only element $\star$. Therefore, for any valuation ρ,

$$|\exists x \,.\, \neg x|_{M,\rho} = |\neg x|_{M,\rho} = M \setminus |x|_{M,\rho} = M \setminus \{\star\} = \emptyset$$

On the other hand, $|\neg\bot|_{M,\rho} = M \setminus \emptyset = M = \{\star\}$. Therefore, $|\neg\bot \to \exists x \,.\, \neg x|_{M,\rho} = M \setminus (\{\star\} \setminus \emptyset) = M \setminus \{\star\} = \emptyset \neq M$. The given instance is not a valid pattern. □

The intuitive reason why ($\exists$-Quantifier') is not necessarily valid is that patterns can be interpreted as any subsets, while element variables (such as x in ($\exists$-Quantifier')) can only be evaluated to individual elements. Therefore, ($\exists$-Quantifier') is valid if ψ is a *functional pattern*. Otherwise, ($\exists$-Quantifier') may be invalid.

($\exists$-Quantifier') where ψ is required to be a functional pattern is reminiscent of the substitution axiom in FOL, which takes the following form:

$$(\text{FOL Substitution}) \qquad \varphi[t/x] \to \exists x \,.\, \varphi$$

where t is a term and φ is a FOL formula. It is possible to capture (FOL SUBSTITUTION) by a similar matching μ-logic pattern after we discuss how functional patterns can be axiomatically defined in matching μ-logic. We leave the discussion to **??**.

In literature where $\forall$ instead of $\exists$ is a primitive construct of the syntax, ($\exists$-QUANTIFIER) is replaced with the following equivalent proof rule:

$$(\forall\text{-Quantifier}) \qquad (\forall x \,.\, \varphi) \to \varphi[y/x]$$

We prove in **??** that ($\forall$-QUANTIFIER) can be proved from ($\exists$-QUANTIFIER).

The proof rule ($\exists$-GENERALIZATION) generalizes the free variables that occur on the left-hand side of an implication pattern. Note that the side condition $x \notin \mathbb{FV}(\varphi_2)$ is necessary to the soundness of the proof rule. The following (counter)example shows that without the side condition, the proof rule has instances that are not valid.

Example 3.56. Let M be any model with more than one elements. Then, the following hold:

1. $M \vDash x \to x$;
2. $M \nvDash (\exists x \,.\, x) \to x$.

It is straightforward to verify that (1) and (2) hold. Let ρ be any valuation. For (1), $|x \to x|_{M,\rho} = M \setminus (\{\rho(x)\} \setminus \{\rho(x)\}) = M \setminus \emptyset = M$, and thus $M \vDash x \to x$. For (2), $|(\exists x \,.\, x) \to x|_{M,\rho} = M \setminus (|\exists x \,.\, x|_{M,\rho} \setminus \{\rho(x)\}) = M \setminus (M \setminus \{\rho(x)\}) = \{\rho(x)\} \neq M$ (noting that $|\exists x \,.\, x|_{M,\rho} = M$, according to **??**). Therefore, $M \nvDash (\exists x \,.\, x) \to x$.

Note that (1) and (2) would constitute an instance of the proof rule ($\exists$-GENERALIZATION) if the side condition $x \notin \mathbb{FV}(\varphi_2)$ were not required. This example shows that the side condition is necessary for the soundness of the proof rule and cannot be eliminated.

We wrap up the discussion by a useful lemma that states that FOL reasoning is generally supported by the matching logic proof system.

Lemma 3.57. *Let $\Psi(p_1, \ldots, p_n)$ be a FOL formula where $p_1, \ldots, p_n$ are atomic predicate symbols. For patterns $\varphi_1, \ldots, \varphi_n$, we write $\Psi(\varphi_1, \ldots, \varphi_n)$ for the result of replacing $p_1, \ldots, p_n$ for the patterns $\varphi_1, \ldots, \varphi_n$. TBA.*

We leave the proof of the lemma to **??**.

Proof Rules that Support Frame Reasoning

The terminology *frame* refers to the infrastructure/context in which reasoning takes place. For example, in artificial intelligence, frame can mean the "common sense" of human beings. In computer science, frame can mean the contexts in which reasoning takes place. For example, consider the following implication between two configurations of a programming language:

$$\left\langle \langle c \rangle_{\text{code}} \, \langle 1 \mapsto 2 * 2 \mapsto 3 \rangle_{\text{heap}} \cdots \right\rangle_{\text{top}} \to \left\langle \langle c \rangle_{\text{code}} \, \langle \mathsf{list}(1) \rangle_{\text{heap}} \cdots \right\rangle_{\text{top}}$$

Frame reasoning allows us to reduce the above implication between two computation configurations into the following *simpler* implication:

$$1 \mapsto 2 * 2 \mapsto 3 \to \mathsf{list}(1)$$

In other words, frame reasoning allows us to reduce *global* reasoning into *local* reasoning, by dropping the context that is common to both sides of the implication. In the above example, the common context is

$$C_{comm} \equiv \left\langle \langle c \rangle_{\mathrm{code}} \, \langle \square \rangle_{\mathrm{heap}} \cdots \right\rangle_{\mathrm{top}}$$

In matching μ-logic, frame reasoning is supported by the three (Propagation) rules and the (Framing) rule. Note that these rules require that the context C be an application context. The following proposition states that frame reasoning is closely related to the pointwise extension of the application function in a matching μ-logic model.

Definition 3.58. Let C be a context. Let M be a model and ρ be a valuation. Under M and ρ, the context C derives a function $C_{M,\rho} \colon \mathcal{P}(M) \to \mathcal{P}(M)$ as follows:

$$C_{M,\rho}(A) = |C|_{M,\rho[A/\square]}, \qquad \text{for } A \subseteq M$$

Lemma 3.59. *For any application context C, the function $C_{M,\rho}$ has the following properties:*

1. $C_{M,\rho}(\emptyset) = \emptyset$;
2. $C_{M,\rho}(A \cup B) = C_{M,\rho}(A) \cup C_{M,\rho}(B)$;
3. $C_{M,\rho}(\bigcup_\lambda A_\lambda) = \bigcup_\lambda C_{M,\rho}(A_\lambda)$;
4. $A \subseteq B$ *implies* $C_{M,\rho}(A) \subseteq C_{M,\rho}(B)$.

Proof. By structural induction on C. □

Lemma 3.59 gives us intuition about the (Propagation) proof rules and the (Framing) rule.

In the proof system, the (Framing) rule requires that C be an application context. However, we can generalize it to *positive contexts*.

Definition 3.60. Let C be a context. We call C a *positive context* if the placeholder variable $\square$ occurs positively in C and $\square$ is not in scope of any fixpoint operators.

Lemma 3.61. *If C is a positive context, then:*

$$\frac{\varphi_1 \to \varphi_2}{C[\varphi_1] \to C[\varphi_2]}$$

Proof Rules that Support Fixpoint Reasoning

In matching μ-logic, fixpoint reasoning is supported via the proof rules (Prefixpoint) and (Knaster-Tarski). To give the semantic intuition behind these two proof rules, we first review the Knaster-Tarski fixpoint theorem **todo**.

Theorem 3.62 (Knaster-Tarski Fixpoint Theorem)**.** *Let M be a nonempty set and $\mathcal{F} \colon \mathcal{P}(M) \to \mathcal{P}(M)$ be a monotone function, i.e., $\mathcal{F}(A) \subseteq \mathcal{F}(B)$ whenever $A \subseteq B$. Then $\mathcal{F}$ has a unique least fixpoint denoted by* **lfp** $\mathcal{F}$ *and a unique greatest fixpoint denoted by* **gfp** $\mathcal{F}$*, such that*

$$\mathbf{lfp}\,\mathcal{F} = \bigcap\{A \subseteq M \mid \mathcal{F}(A) \subseteq A\} \qquad\qquad \mathbf{gfp}\,\mathcal{F} = \bigcup\{A \subseteq M \mid A \subseteq \mathcal{F}(A)\}$$

The following lemma is a direct corollary of Theorem 3.62.

Lemma 3.63. *Let M be a nonempty set and $\mathcal{F} \colon \mathcal{P}(M) \to \mathcal{P}(M)$ be a monotone function. Then, the following two propositions hold:*

1. $\mathcal{F}(\mathbf{lfp}\,\mathcal{F}) = \mathbf{lfp}\,\mathcal{F}$*;*
2. $\mathbf{lfp}\,\mathcal{F} \subseteq A$ *whenever* $\mathcal{F}(A) \subseteq A$*.*

The proof rules (Prefixpoint) and (Knaster-Tarski) are the logical incarnation of the two propositions in Lemma 3.63.

Miscellaneous Proof Rules

3.3.2 Soundness Theorem

Theorem 3.64 (Soundness Theorem)**.** *For any* Γ *and* φ, $\Gamma \vdash \varphi$ *implies* $\Gamma \vDash \varphi$.

3.3.3 Basic Matching Logic Reasoning

To do Talk about formal reasoning (for all theories). (**??**)

1. Propositional reasoning and FOL reasoning.
2. Basic fixpoint reasoning.