Matching Logic: A Logic for Structural Reasoning

Grigore Roşu
University of Illinois at Urbana-Champaign
grosu@illinois.edu

***Abstract*—Matching logic is a first-order logic (FOL) variant to reason about structure. Its sentences, called *patterns*, are constructed using *variables*, *symbols*, *connectives* and *quantifiers*, but no difference is made between function and predicate symbols. In models, a pattern evaluates into a power-set domain (the set of values that *match* it), in contrast to FOL where functions, predicates and connectives map into a domain. Matching logic generalizes several logical frameworks important for program analysis, such as: propositional logic, algebraic specification, FOL with equality, and separation logic. Patterns allow for specifying separation requirements at any level in any program configuration, not only in the heaps or stores, without any special logical constructs for that: the very nature of pattern matching is that if two structures are matched as part of a pattern, then they can only be spatially separated. Like FOL, matching logic can also be translated into pure predicate logic with equality, but it also admits its own sound and complete proof system.**

I. Introduction and Motivation

Matching logic's formulae, called *patterns*, are defined using variables, symbols from a signature, and FOL connectives and quantifiers. We only treat the many-sorted first-order case in this paper, but the same ideas can be extended to order-sorted or higher-order contexts. Specifically, if (S, Σ) is a many-sorted signature and Var an S-sorted set of variables, the patterns φ_s of sort $s \in S$ are defined as follows:

$$\begin{array}{rcl} \varphi_s & ::= & x \in Var_s \;\mid\; \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) \quad \text{where } \sigma \in \Sigma_{s_1\ldots s_n, s} \\ & \mid & \neg\varphi_s \;\mid\; \varphi_s \wedge \varphi_s \;\mid\; \exists x.\varphi_s \qquad \text{where } x \in Var \end{array}$$

Semantically, a *model* M consists of a carrier M_s for each sort s, same like in FOL, but interprets symbols $\sigma \in \Sigma_{s_1\ldots s_n,s}$ as maps $\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$ associating a *set* of elements to any tuple of arguments. In particular, σ_M can be a function, when the result sets contain only one element, or a partial function, when the result sets contain at most one element. Any M-*valuation* $\rho : Var \to M$ extends to a map $\overline{\rho}$ taking patterns to sets of values, where $\neg$ is interpreted as complement, $\wedge$ as intersection, and $\exists$ as union over all compatible valuations. If φ is a pattern and $a \in \overline{\rho}(\varphi)$ then we say that *a matches* φ (*with* ρ). A pattern φ is *valid in M* iff $\overline{\rho}(\varphi) = M$ (i.e., it is matched by all elements of M), and it is *valid* iff it is valid in all models. It turns out that, unlike in FOL, equality can be defined in matching logic (Section III-C): i.e., $\varphi_1 = \varphi_2$ can be defined as a pattern so that, given any model M and any M-valuation $\rho : Var \to M$, $\varphi_1 = \varphi_2$ is either matched by all elements when $\overline{\rho}(\varphi_1) = \overline{\rho}(\varphi_2)$, or by none otherwise.

For example, if Σ is the signature of Peano natural numbers and M is the model of natural numbers with 0 and *succ* interpreted accordingly, then the pattern $\exists x \,.\, succ(x)$ is matched by all positive numbers. If we want to only allow models in which 0 and *succ* are total functions and *succ* is injective, and whose elements are either zero or successors of other elements, then we add the following axioms:

$$\begin{array}{l} \exists y \,.\, 0 = y \\ \exists y \,.\, succ(x) = y \\ succ(x_1) = succ(x_2) \to x_1 = x_2 \\ 0 \vee \exists x \,.\, succ(x) \end{array}$$

where $\vee$ and $\to$ are defined as usual: $\varphi_1 \vee \varphi_2$ is $\neg(\neg\varphi_1 \wedge \neg\varphi_2)$ and $\varphi_1 \to \varphi_2$ is $\neg(\varphi_1 \wedge \neg\varphi_2)$. We can go further and axiomatize *plus* the same way we are used to in algebraic specification:

$$\begin{array}{l} plus(0, y) = y \\ plus(succ(x), y) = succ(plus(x, y)) \end{array}$$

or equivalently as the following equality pattern:

$$plus(x, y) = (x = 0 \wedge y \vee \exists z \,.\, x = succ(z) \wedge succ(plus(z, y)))$$

Let us next define a matching logic specification whose symbols are not all functions anymore. Consider a typical algebraic specification of maps from natural to integer numbers, with *emp* the empty map, $\_ \mapsto \_$ constructing a map of one binding and $\_ \mapsto [\_]$ constructing a map of consecutive bindings, and $\_ * \_$ the partial function merging two maps. In addition to the usual unit, associativity and commutativity axioms for *emp* and $\_ * \_$, we also add the following pattern axioms:

$$\begin{array}{l} \neg(0 \mapsto a) \\ (x \mapsto a * y \mapsto b) \to x \neq y \\ x \mapsto [\epsilon] = emp \\ x \mapsto [a, S] = x \mapsto a * (x + 1) \mapsto [S] \end{array}$$

We purposely chose a syntax for maps which resembles that used in separation logic [8]–[10], to ease the comparison. The first pattern states that 0 cannot serve as the key of any binding, the second that the keys of different bindings in a map must be distinct, and the last two desugar the consecutive binding construct. Consider now a symbol $list \in \Sigma_{Nat \times Seq, Map}$ taking a natural number x and a sequence of integers S to a set of maps $list(x, S)$, together with the following two pattern axioms

$$\begin{array}{l} list(0, \epsilon) = emp \\ list(x, n \cdot S) = \exists z \,.\, x \mapsto [n, z] * list(z, S) \end{array}$$

which look similar to how the list predicate is defined in separation logics using recursive predicates, although note that in matching logic there is no predicate, let alone recursive predicates. In matching logic, the equations above use the same principle to define *list* as the Peano equations did to define *plus*: pattern equations. We can now show (see Section IV) that in the model whose *Map* carrier consists of the finite-domain partial maps and $\mapsto$ and $*$ are interpreted appropriately, the interpretation of $list(x, S)$ is precisely the set of all singly-linked lists starting with $x \neq 0$ and comprising the sequence of integers S. That is, the $list(x, S)$ pattern is matched by precisely the desired lists in the intended model. Using the generic proof system of matching logic in Section V, we can now derive properties about lists in this specification, such as, for example, $(1 \mapsto 5 * 2 \mapsto 0 * 7 \mapsto 9 * 8 \mapsto 1) \to list(7, 9 \cdot 5)$:

$$\begin{array}{rl} & 1 \mapsto 5 * 2 \mapsto 0 * 7 \mapsto 9 * 8 \mapsto 1 \\ = & 1 \mapsto [5,0] * 7 \mapsto [9,1] \\ = & 1 \mapsto [5,0] * list(0,\epsilon) * 7 \mapsto [9,1] \\ \to & (\exists z\,.\,1 \mapsto [5,z] * list(z,\epsilon)) * 7 \mapsto [9,1] \\ = & list(1,5) * 7 \mapsto [9,1] \\ \to & \exists z\,.\,7 \mapsto [9,z] \wedge list(z,5) \\ = & list(7, 9 \cdot 5) \end{array}$$

Section IV shows that separation logic is equivalent to the matching logic theory above, in the finite-domain map model.

The benefits of matching logic can be perhaps best noticed when there are no immediate existing logics that we can use to reason about structure. Consider, for example, the operational semantics of a real language like C, whose configuration can be defined with ordinary algebraic specification but has more than 70 semantic cells [5]. The semantic cells can be nested, their grouping is associative and commutative, and are written using symbols $\langle ... \rangle_{\mathsf{cell}}$. There is a top cell called $\langle ... \rangle_{\mathsf{cfg}}$, holding a subcell $\langle ... \rangle_{\mathsf{heap}}$ among many others. We can then globalize the local reasoning above to the entire C configuration proving the following property using the same proof system in Section V:

$$\forall c : Cfg . \forall h : Map\,.\,(\langle\langle 1 \mapsto 5 * 2 \mapsto 0 * 7 \mapsto 9 * 8 \mapsto 1 * h\rangle_{\mathsf{heap}}\ c\rangle_{\mathsf{cfg}} \\ \to \langle\langle list(7, 9 \cdot 5) * h\rangle_{\mathsf{heap}}\ c\rangle_{\mathsf{cfg}})$$

Note that quantification over the heap or over the configuration are still first-order in matching logic, since the heap or fragments of it such as a singly-linked list, are terms/patterns of sort *Map* (and not predicates, like in separation logic). We refer to such variables like h and c matching the remaining contents of a cell "cell" as *(structural) cell frames*; e.g., h is the *heap frame* and c is the *configuration frame*, and write them as ellipses "..." when their particular name is irrelevant.

The operational semantics of C consists of more than 1,200 rewrite rules between matching logic patterns (ordinary terms with variables are particular patterns). Matching logic reasoning can be used in-between semantic steps to re-arrange the configuration so that semantic rules match or assertions can be proved, yielding a sound and complete verification framework based on symbolic execution using the language semantics, but without employing any language-specific logic. This novel verification approach is discussed in [11], [12]. Here we only show one example using MatchC [11], a program verifier for C based on an early variant of matching logic discussed shortly.

Fig. 1 shows a C function with specifications (in grey), which can be automatically verified by MatchC[1]. It reads n elements

```
struct listNode { int val; struct listNode *next; };

void list_read_write(int n)
 rule   ⟨$ ⇒ return;  ···⟩k ⟨A ⇒ ·  ···⟩in ⟨···  · ⇒ rev(A)⟩out
 requires   n = len(A)
{ int i=0;
  struct listNode *x=0;
  inv   ⟨?B  ···⟩in ⟨···  list(x, ?A)  ···⟩heap
        ∧ i ≤ n  ∧  len(?B) = n − i  ∧  A = rev(?A)@?B
  while (i < n) {
    struct listNode *y = x;
    x = (struct listNode*) malloc(sizeof(struct listNode));
    scanf("%d", &(x->val));
    x->next = y;
    i += 1;
  }
  inv   ⟨···  ?A⟩out ⟨···  list(x, ?B)  ···⟩heap  ∧  rev(A) = ?A@?B
  while (x) {
    struct listNode *y;
    y = x->next;
    printf("%d␣",x->val);
    free(x);
    x = y;
  }
}
```

Fig. 1. Reading, storing, and reverse writing a sequence of integers

from the standard input and writes them to the standard output in reverse order. Internally, it stores the elements into a singly-linked list, which is allocated as the elements are read. Then it outputs the elements from the list, which is deallocated as the elements are written. In the end, the heap stays unchanged (implicitly stated, because the heap does not appear in the function specification). The rule specification of the function states its semantics/summary: the body ($) returns in the code cell $\langle\rangle_{\mathsf{k}}$ possibly followed by other code (as mentioned, "..." are structural frames, that is, universally quantified "anonymous" variables), the sequence A of size n is consumed from the prefix of the input buffer (A is rewritten to "·", the unit of sequences, possibly followed by more input), and the reversed sequence rev(A) is put at the end of the output buffer.

The invariant specification of the first loop states that the pattern list(x, ?A) can be matched somewhere in the heap, and that the sequence ?B of size n − i is available in the input buffer such that A is the reverse of the sequence that x points to, rev(?A), concatenated with ?B. By convention, Boolean patterns like i ≤ n can be used in any context and they are either matched by all elements when they hold, or by no elements when they don't hold (Section III-E). The variables starting with a ? are assumed existentially quantified. The invariant of the second loop says that a sequence ?A can be matched as a suffix of the output buffer and sequence ?B can be matched within a list that x points to in the heap, such that the ?A@B is the reverse of the original sequence A. The verification of this function consists of executing the operational semantics of C symbolically on all five paths starting with the left-hand-side of the function specification rule, and each time a pattern is encountered to defer a pattern implication proof task to the matching logic prover. For example, the last proof task is:

$$\exists ?\mathsf{A}\,.\,\exists ?\mathsf{B}\,.\,\langle\langle I\rangle_{\mathsf{in}}\ \langle O, ?\mathsf{A}\rangle_{\mathsf{out}}\ \langle \mathsf{list}(\mathsf{x}, ?\mathsf{B}) * H\rangle_{\mathsf{heap}}\ C\rangle_{\mathsf{cfg}} \\ \wedge\ \mathsf{rev}(\mathsf{A}) = ?\mathsf{A}@?\mathsf{B} \wedge \mathsf{x} = 0 \\ \to \langle\langle I\rangle_{\mathsf{in}}\ \langle O, \mathsf{rev}(\mathsf{A})\rangle_{\mathsf{out}}\ \langle H\rangle_{\mathsf{heap}}\ C\rangle_{\mathsf{cfg}}$$

[1]The reader can check it, as well as dozens of other programs including Schorr-Waite, using the online MatchC interface at http://matching-logic.org; this function is under the io folder and it takes about 150ms to verify.

which can be easily proved using the matching logic proof system in Section V and the given pattern axioms.

This is the first paper introducing matching logic in its full generality. An earlier simpler variant was introduced as a state specification logic in the context of larger verification works during the last four years (we only mention the first [13] and the last [12]), and implemented in MatchC [11] by reduction to Maude [3] (for matching) and to Z3 [4] (for domain constraints). However, that variant shares only the basic intuition of "terms as formulae" with the logic presented in this paper, and was only syntactic sugar for first-order logic (FOL) with equality in a fixed model, essentially allowing only term patterns t and regarding them as syntactic sugar for equalities $\square = t$.

Matching logic builds upon intuitions from and relates to at least three important logical frameworks: (1) *Relation algebra (RA)* (see, e.g., [14]), noticing that our interpretations of symbols as functions to powersets are equivalent to relations; although our interpretation of symbols captures better the intended meaning of pattern and matching, and our proof system is quite different from that of RA, like with FOL we expect a tight relationship between matching logic and RA, which is left as future work; (2) *Partial FOL* (see, e.g., [6] for a recent work and a survey), noticing that our interpretations of symbols into powersets are more general than partial functions (Section III-C shows how we define definedness); and (3) *Separation logics (SL)* (see, e.g., [8]), discussed above.

Section II introduces the syntax and semantics of matching logic, and shows that, like FOL with equality, it also translates to predicate logic. Section III enumerates a series of examples. Section IV discusses the important case of maps, and shows how separation logic can be framed as a matching logic theory. Finally, Section V introduces a sound and complete proof system for matching logic, and Section VI concludes.

II. Matching Logic

We assume the reader familiar with many-sorted sets, functions, and FOL. For any given set of sorts S, we assume *Var* is an S-sorted set of variables, sortwise infinite and disjoint. We may write $x : s$ instead of $x \in Var_s$; when the sort of x is irrelevant, we just write $x \in Var$. We let $\mathcal{P}(M)$ denote the powerset of a many-sorted set M, which is itself many-sorted.

A. Basic Definitions

Definition 1: Let (S, Σ) be a many-sorted signature of *symbols*. Matching logic *formulae*, also called *patterns*, of all sorts $s \in S$ are inductively defined as follows:

$$\begin{array}{rcl} \varphi_s & ::= & x \in Var_s \quad | \quad \sigma(\varphi_{s_1}, \ldots, \varphi_{s_n}) \quad \text{where } \sigma \in \Sigma_{s_1 \ldots s_n, s} \\ & | & \neg\varphi_s \quad | \quad \varphi_s \wedge \varphi_s \quad | \quad \exists x.\varphi_s \qquad \text{where } x \in Var \end{array}$$

Let Pattern be the S-sorted set of matching logic formulae. By abuse of language, we refer to the symbols in Σ also as patterns; we can think of $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ as the pattern $\sigma(x_1 : s_1, \ldots, x_n : s_n)$.

To compact notation, $\varphi \in$ Pattern means φ is any pattern, while $\varphi_s \in$ Pattern or $\varphi \in$ Pattern$_s$ that it has sort s.

We adopt the following derived constructs:

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

and let $FV(\varphi)$ denote the *free variables* of φ, defined as usual.

Definition 2: A *matching logic* (S, Σ)*-model* M, or simply a *model* when (S, Σ) is understood, consists of

- An S-sorted set $\{M_s\}_{s \in S}$, where each set M_s, called the *carrier of sort* s *of* M, is assumed non-empty; and
- A function $\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \rightarrow \mathcal{P}(M_s)$ for each symbol $\sigma \in \Sigma_{s_1 \ldots s_n, s}$, called the *interpretation* of σ in M.

Note that usual (S, Σ)-algebras are special cases of matching logic models, where $|\sigma_M(m_1, \ldots, m_n)| = 1$ for any $m_1 \in M_{s_1}$, $\ldots$, $m_n \in M_{s_n}$. Similarly, partial (S, Σ)-algebras also fall as special cases, where $|\sigma_M(m_1, \ldots, m_n)| \leq 1$, because we can capture undefinedness of σ_M in m_1, $\ldots$, m_n with $\sigma_M(m_1, \ldots, m_n) = \emptyset$.

We tacitly use the same notation σ_M for its extension $\mathcal{P}(M_{s_1}) \times \cdots \times \mathcal{P}(M_{s_n}) \rightarrow \mathcal{P}(M_s)$ to sets of arguments, i.e., $\sigma_M(A_1, \ldots, A_n) = \bigcup\{\sigma_M(a_1, \ldots, a_n) \mid a_1 \in A_1, \ldots, a_n \in A_n\}$, where $A_1 \subseteq M_{s_1}, \ldots, A_n \subseteq M_{s_n}$.

Definition 3: Given a model M and a map $\rho : Var \rightarrow M$, called M*-valuation*, let its extension $\overline{\rho}$: Pattern $\rightarrow \mathcal{P}(M)$ be inductively defined as follows:

- $\overline{\rho}(x) = \{\rho(x)\}$, for all $x \in Var_s$
- $\overline{\rho}(\sigma(\varphi_{s_1}, \ldots, \varphi_{s_n})) = \sigma_M(\overline{\rho}(\varphi_1), \ldots \overline{\rho}(\varphi_n))$
- $\overline{\rho}(\neg\varphi_s) = M_s \setminus \overline{\rho}(\varphi_s)$ (where "$\setminus$" is the set difference)
- $\overline{\rho}(\varphi_1 \wedge \varphi_2) = \overline{\rho}(\varphi_1) \cap \overline{\rho}(\varphi_2)$
- $\overline{\rho}(\exists x.\varphi) = \bigcup\{\overline{\rho'}(\varphi) \mid \rho' : Var \rightarrow M,\ \rho'\restriction_{Var\setminus\{x\}} = \rho\restriction_{Var\setminus\{x\}}\}$

The extension of ρ works as expected with the derived constructs. From here on we tacitly use the following properties:

- $\overline{\rho}(\bot_s) = \emptyset$ and $\overline{\rho}(\top_s) = M_s$
- $\overline{\rho}(\varphi_1 \vee \varphi_2) = \overline{\rho}(\varphi_1) \cup \overline{\rho}(\varphi_2)$
- $\overline{\rho}(\varphi_1 \rightarrow \varphi_2) = \{m \in M_s \mid m \in \overline{\rho}(\varphi_1) \text{ implies } m \in \overline{\rho}(\varphi_2)\} = M_s \setminus (\overline{\rho}(\varphi_1) \setminus \overline{\rho}(\varphi_2))$
- $\overline{\rho}(\varphi_1 \leftrightarrow \varphi_2) = \{m \in M_s \mid m \in \overline{\rho}(\varphi_1) \text{ iff } m \in \overline{\rho}(\varphi_2)\} = M_s \setminus (\overline{\rho}(\varphi_1) \,\Delta\, \overline{\rho}(\varphi_2))$ ("Δ" is set symmetric difference)
- $\overline{\rho}(\forall x.\varphi) = \bigcap\{\overline{\rho'}(\varphi) \mid \rho' : Var \rightarrow M,\ \rho'\restriction_{Var\setminus\{x\}} = \rho\restriction_{Var\setminus\{x\}}\}$

Definition 4: Model M *satisfies* pattern φ_s, written $M \models \varphi_s$, if and only if $\overline{\rho}(\varphi_s) = M_s$ for all $\rho : Var \rightarrow M$.

Proposition 1: The following properties hold:

- If $\rho_1, \rho_2 : Var \rightarrow M$, $\rho_1\restriction_{FV(\varphi)} = \rho_2\restriction_{FV(\varphi)}$ then $\overline{\rho_1}(\varphi) = \overline{\rho_2}(\varphi)$
- If $x \in Var_s$ then $M \models x$ iff $|M_s| = 1$
- If $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ and $\varphi_1, \ldots, \varphi_n$ are patterns of sorts $s_1, \ldots, s_n$, respectively, then $M \models \sigma(\varphi_1, \ldots, \varphi_n)$ iff $\sigma_M(\overline{\rho}(\varphi_1), \ldots \overline{\rho}(\varphi_n)) = M_s$ for any $\rho : Var \rightarrow M$
- $M \models \neg\varphi$ iff $\overline{\rho}(\varphi) = \emptyset$ for any $\rho : Var \rightarrow M$
- $M \models \varphi_1 \wedge \varphi_2$ iff $M \models \varphi_1$ and $M \models \varphi_2$
- If $\exists x.\varphi_s$ is closed, then $M \models \exists x.\varphi_s$ iff $\bigcup\{\overline{\rho}(\varphi_s) \mid \rho : Var \rightarrow M\} = M_s$; in particular, $M \models \exists x.x$
- $M \models \varphi_1 \rightarrow \varphi_2$ iff $\overline{\rho}(\varphi_1) \subseteq \overline{\rho}(\varphi_2)$ for all $\rho : Var \rightarrow M$
- $M \models \varphi_1 \leftrightarrow \varphi_2$ iff $\overline{\rho}(\varphi_1) = \overline{\rho}(\varphi_2)$ for all $\rho : Var \rightarrow M$
- $M \models \forall x.\varphi$ iff $M \models \varphi$

Note that property "if φ closed then $M \models \neg\varphi$ iff $M \not\models \varphi$", which holds in FOL, does not hold in matching logic. Indeed, suppose φ is a constant symbol, say 0, of sort s. Then $M \models \neg 0$ is equivalent to $0_M = \emptyset$, while $M \not\models 0$ is equivalent to $0_M \neq M_s$.

Definition 5: Pattern φ is *valid*, written $\models \varphi$, iff $M \models \varphi$ for all M. If $F \subseteq$ PATTERN then $M \models F$ iff $M \models \varphi$ for all $\varphi \in F$. F *entails* φ, written $F \models \varphi$, iff $M \models F$ implies $M \models \varphi$. A *matching logic specification* is a triple (S, Σ, F) with $F \subseteq$ PATTERN.

B. Reduction to Predicate Logic

It is well-known that FOL formulae (with function symbols) can be translated into equivalent predicate logic formulae, by replacing each function symbol with a predicate symbol and then systematically transforming terms into formulae. We can similarly translate patterns into equivalent predicate logic formulae. Consider pure predicate logic with equality and no constant symbols, whose satisfaction relation is written $\models^{=}_{PL}$. If (S, Σ) is a matching logic signature, let (S, Π_Σ) be the predicate logic signature with $\Pi_\Sigma = \{\pi_\sigma : s_1 \times \cdots \times s_n \times s \mid \sigma \in \Sigma_{s_1 \ldots s_n, s}\}$. We define the translation PL of matching logic (S, Σ)-patterns into predicate logic (S, Π_Σ)-formulae inductively as follows:

$$PL(\varphi) = \forall r \,.\, PL(\varphi, r)$$

$$\begin{aligned}
PL(x, r) &= (x = r) \\
PL(\sigma(\varphi_1, \ldots, \varphi_n), r) &= \exists r_1 \cdots \exists r_n \,.\, PL(\varphi_1, r_1) \wedge \cdots \wedge PL(\varphi_n, r_n) \\
&\qquad \wedge \pi_\sigma(r_1, \ldots, r_n, r) \\
PL(\neg\varphi, r) &= \neg PL(\varphi, r) \\
PL(\varphi_1 \wedge \varphi_2, r) &= PL(\varphi_1, r) \wedge PL(\varphi_2, r) \\
PL(\exists x \,.\, \varphi, r) &= \exists x \,.\, PL(\varphi, r)
\end{aligned}$$

$$PL(\{\varphi_1, \ldots, \varphi_n\}) = \{PL(\varphi_1), \ldots, PL(\varphi_n)\}$$

Then the following result holds, same like for FOL:

Proposition 2: $F \models \varphi$ iff $PL(F) \models^{=}_{PL} PL(\varphi)$

Proof: It suffices to show that there is a bijective correspondence between matching logic (S, Σ)-models M and predicate logic (S, Π_Σ)-models M', such that $M \models \varphi$ iff $M' \models^{=}_{PL} PL(\varphi)$ for any (S, Σ)-pattern φ. The bijection is defined as follows:

- $M'_s = M_s$ for each sort $s \in S$;
- $\pi_{\sigma M'} \subseteq M_{s_1} \times \cdots \times M_{s_n} \times M_s$ with $(a_1, \ldots, a_n, a) \in \pi_{\sigma M'}$ iff $\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$ with $a \in \sigma_M(a_1, \ldots, a_n)$.

To show $M \models \varphi$ iff $M' \models^{=}_{PL} PL(\varphi)$, it suffices to show $a \in \overline{\rho}(\varphi)$ iff $\rho[a/r] \models^{=}_{PL} PL(\varphi, r)$ for any $\rho : Var \to M$, which follows easily by structural induction on φ. ■

Proposition 2 gives a sound and complete procedure for matching logic reasoning: translate the specification (S, Σ, F) and pattern to prove φ into the predicate logic specification $(S, \Pi_\Sigma, PL(F))$ and formula $PL(\varphi)$, respectively, and then derive it using the sound and complete proof system of predicate logic. However, translating patterns to predicate logic formulae makes reasoning harder not only for humans, but also for machines, since new quantifiers are introduced. For example, $(1 \mapsto 5 * 2 \mapsto 0 * 7 \mapsto 9 * 8 \mapsto 1) \to list(7, 9 \cdot 5)$ discussed and proved in Section I, translates into the formula (to keep it small, we do not translate the numbers) $\forall r \,.\, (\exists r_1 \,.\, \exists r_2 \,.\, \pi_{\mapsto}(1, 5, r_1) \wedge (\exists r_3 \,.\, \exists r_4 \,.\, \pi_{\mapsto}(2, 0, r_3) \wedge (\exists r_5 \,.\, \exists r_6 \,.\, \pi_{\mapsto}(7, 9, r_5) \wedge \pi_{\mapsto}(8, 1, r_6) \wedge \pi_*(r_5, r_6, r_4)) \wedge \pi_*(r_3, r_4, r_2)) \wedge \pi_*(r_1, r_2, r)) \to \exists r_7 \,.\, \pi_{\cdot}(9, 5, r_7) \wedge \pi_{list}(7, r_7, r)$. What we would like is to reason directly with matching logic patterns, the same way we reason directly with terms in FOL without translating them to predicate logic.

Proposition 3: The following hold for matching logic:

1) $\models \varphi$, where φ is a propositional tautology (over patterns)
2) Modus ponens: $\models \varphi_1$ and $\models \varphi_1 \to \varphi_2$ implies $\models \varphi_2$
3) $\models (\forall x \,.\, \varphi_1 \to \varphi_2) \to (\varphi_1 \to \forall x \,.\, \varphi_2)$ when $x \notin FV(\varphi_1)$
4) Universal generalization: $\models \varphi$ implies $\models \forall x \,.\, \varphi$

Proposition 3 states that the proof system of pure predicate logic is actually sound for matching logic *as is*, so we do not need to translate patterns to predicate logic formulae in order to reason about them. Section V will show that a few additional proof rules yield a sound and complete proof system for matching logic, in a similar vein to how the Substitution rule $(\forall x \,.\, \varphi \to \varphi[t/x])$ together with the four proof rules of pure predicate logic yield a sound and complete proof system for FOL. But before that, we demonstrate the usefulness of matching logic by a series of examples and applications.

III. EXAMPLES AND NOTATIONS

We have already seen some simple patterns in Section II, such as $\exists x.x$ which is satisfied by all models, and $\forall x.x$ which is satisfied only by models whose carrier of the sort of x contains only one element. In this section we illustrate matching logic by means of a series of more complex examples.

A. Propositional logic

If we take S to contain only one sort, *Prop*, Σ to be empty, and drop the existential quantifier, then the syntax of matching logic becomes that of propositional calculus:

$$\varphi ::= Var_{Prop} \mid \neg\varphi \mid \varphi \wedge \varphi$$

The following expected result holds:

Proposition 4: For any proposition φ, $\models_{Prop} \varphi$ iff $\models \varphi$.

An alternative way to capture propositional calculus is to add a constant symbol to Σ for each propositional variable, and then associate a ground pattern to each proposition. Proposition 4 still holds, despite the fact that propositional constants can be interpreted as arbitrary sets. That is because $(\mathcal{P}(M), \neg_M, \cap)$ is a model of propositional logic for any set M.

B. Pure predicate logic

If S is a sort set and Π is a set of predicate symbols, the syntax of pure predicate logic formulae (without equality) is

$$\begin{array}{lcl}
\varphi & ::= & \pi(x_1, \ldots, x_n) \;\text{ where } \pi \in \Pi_{s_1 \ldots s_n} \\
& \mid & \neg\varphi \quad \mid \quad \varphi \wedge \varphi \quad \mid \quad \exists x.\varphi
\end{array}$$

We can pick a new sort name, *Pred*, and construct a matching logic signature $(S \cup \{Pred\}, \Sigma)$ where $\Sigma_{s_1 \ldots s_n, Pred} = \Pi_{s_1 \ldots s_n}$. Then any predicate logic formula can be trivially regarded as a matching logic pattern. The following result then holds:

Proposition 5: For any predicate formula φ, $\models_{PL} \varphi$ iff $\models \varphi$.

C. Definedness, Equality, Membership

Pattern definedness, equality and membership can be defined in matching logic, without any special support or logic extensions. Before we do so, it is insightful to understand why we cannot use $\leftrightarrow$ as an equality. Indeed, since $M \models \varphi_1 \leftrightarrow \varphi_2$ iff $\overline{\rho}(\varphi_1) = \overline{\rho}(\varphi_2)$ for all $\rho : Var \to M$, one may be tempted to blindly adopt $\leftrightarrow$ as equality everywhere. For example, given a signature with one sort and one unary symbol f, one may think that the following matching logic specification captures precisely the models in which f is interpreted as a function:

$$\exists y \,.\, f(x) \leftrightarrow y$$

Unfortunately, there are models of the above specification in which the interpretation of f is not a function. Consider, for example, a model M with $M = \{1, 2\}$ and with f_M defined as $f_M(1) = \{1, 2\}$ and $f_M(2) = \emptyset$. Let $\rho : Var \to M$. If $\rho(x) = 1$ then $\overline{\rho}(\exists y \,.\, f(x) \leftrightarrow y) = (M \setminus (\{1, 2\} \,\Delta\, \{1\})) \cup (M \setminus (\{1, 2\} \,\Delta\, \{2\})) = \{1, 2\} = M$. If $\rho(x) = 2$ then $\overline{\rho}(\exists y \,.\, f(x) \leftrightarrow y) = (M \setminus (\emptyset \,\Delta\, \{1\})) \cup (M \setminus (\emptyset \,\Delta\, \{2\})) = \{1, 2\} = M$. Therefore, $M \models \exists y \,.\, f(x) \leftrightarrow y$.

The problem above was that the interpretation of $\varphi_1 \leftrightarrow \varphi_2$ is not equivalent to either $\top$ or $\bot$, as we are used to think in FOL. Specifically, $\overline{\rho}(\varphi_1) \neq \overline{\rho}(\varphi_2)$ does not suffice for $\overline{\rho}(\varphi_1 \leftrightarrow \varphi_2) = \emptyset$ to hold. Indeed, $\overline{\rho}(\varphi_1 \leftrightarrow \varphi_2) = M \setminus (\overline{\rho}(\varphi_1) \,\Delta\, \overline{\rho}(\varphi_2))$ and there is nothing to prevent, e.g., $\overline{\rho}(\varphi_1) \cap \overline{\rho}(\varphi_2) \neq \emptyset$, in which case $\overline{\rho}(\varphi_1) \,\Delta\, \overline{\rho}(\varphi_2) \neq M$. What we would like to have is a proper equality, $\varphi_1 = \varphi_2$, which behaves like a predicate: $\overline{\rho}(\varphi_1 = \varphi_2) = \emptyset$ when $\overline{\rho}(\varphi_1) \neq \overline{\rho}(\varphi_2)$, and $\overline{\rho}(\varphi_1 = \varphi_2) = M$ when $\overline{\rho}(\varphi_1) = \overline{\rho}(\varphi_2)$. Moreover, we want equalities to be used with terms of any sort, and in contexts of any sort.

The above can be achieved methodologically in matching logic, by adding to the signature a *definedness* symbol $[\_]_{s_1}^{s_2} \in \Sigma_{s_1, s_2}$ for any sorts s_1 and s_2, together with the pattern axiom

$$[x \!:\! s_1]_{s_1}^{s_2}$$

The axiom enforces $([\_]_{s_1}^{s_2})_M(m_1) = M_{s_2}$ in all models M for all $m_1 \in M_{s_1}$, which means that for any $\rho : Var \to M$, $\overline{\rho}([\varphi]_{s_1}^{s_2})$ is either $\emptyset$ when $\overline{\rho}(\varphi) = \emptyset$ (i.e., φ undefined in ρ), or is M_{s_2} when $\overline{\rho}(\varphi) \neq \emptyset$ (i.e., φ defined). We can now use $\_ =_{s_1}^{s_2} \_$ and $\_ \in_{s_1}^{s_2} \_$, respectively, as aliases for the following patterns:

$$\begin{array}{rcll} \varphi =_{s_1}^{s_2} \varphi' & \equiv & \neg[\neg(\varphi \leftrightarrow \varphi')]_{s_1}^{s_2} & \text{where } \varphi, \varphi' \in \text{PATTERN}_{s_1} \\ x \in_{s_1}^{s_2} \varphi & \equiv & [x \wedge \varphi]_{s_1}^{s_2} & \text{where } x \in Var_{s_1}, \varphi \in \text{PATTERN}_{s_1} \end{array}$$

Proposition 6: With the above, the following hold:

1) $\overline{\rho}(\varphi =_{s_1}^{s_2} \varphi') = \emptyset$ iff $\overline{\rho}(\varphi) \neq \overline{\rho}(\varphi')$
2) $\overline{\rho}(\varphi =_{s_1}^{s_2} \varphi') = M_{s_2}$ iff $\overline{\rho}(\varphi) = \overline{\rho}(\varphi')$
3) $\models \varphi =_{s_1}^{s_2} \varphi'$ iff $\models \varphi \leftrightarrow \varphi'$
4) $\overline{\rho}(x \in_{s_1}^{s_2} \varphi) = \emptyset$ iff $\rho(x) \notin \overline{\rho}(\varphi)$
5) $\overline{\rho}(x \in_{s_1}^{s_2} \varphi) = M_{s_2}$ iff $\rho(x) \in \overline{\rho}(\varphi)$
6) $\models (x \in_{s_1}^{s_2} \varphi) =_{s_2}^{s_3} (x \wedge \varphi =_{s_1}^{s_2} x)$

From now on we assume equality and membership in matching logic specifications, without mentioning the constructions above. Moreover, since s_1 and s_2 can usually be inferred from context, we write $[\_]$, $=$ and $\in$ instead of $[\_]_{s_1}^{s_2}$, $=_{s_1}^{s_2}$, and $\in_{s_1}^{s_2}$, respectively. If the sort decorations cannot be inferred from context, then we assume the stated property/axiom/rule holds for all such sorts. For example, the generic pattern axiom "$[x]$ where $x \in Var$" replaces all the axioms $[x \!:\! s_1]_{s_1}^{s_2}$ above for the definedness symbol, for all the sorts s_1 and s_2. Similarly, the axiom in Section IV-A defining list patterns within maps,

$$list(x) = (x = 0 \wedge emp \vee \exists z \,.\, x \mapsto z * list(z))$$

is equivalent to the explicit axioms (for all sorts s):

$$list(x) =_{Map}^{s} (x =_{Nat}^{Map} 0 \wedge emp \vee \exists z \,.\, x \mapsto z * list(z))$$

Proposition 3 showed that four of the proof rule/axiom schemas of FOL are already sound for matching logic. Below we show the soundness of several other rule/axiom schemas, essentially proving the soundness of the matching logic proof system, except one rule, Substitution, deferred to Section V:

Proposition 7: The following hold:

1) Equality introduction: $\models \varphi = \varphi$
2) Equality elimination: $\models \varphi_1 = \varphi_2 \wedge \varphi[\varphi_1/x] \to \varphi[\varphi_2/x]$
3) $\models \forall x \,.\, x \in \varphi$ iff $\models \varphi$
4) $\models (x \in y) = (x = y)$ when $x, y \in Var$
5) $\models (x \in \neg\varphi) = \neg(x \in \varphi)$
6) $\models (x \in \varphi_1 \wedge \varphi_2) = (x \in \varphi_1) \wedge (x \in \varphi_2)$
7) $\models (x \in \exists y . \varphi) = \exists y . (x \in \varphi)$, with x and y distinct
8) $\models x \in \sigma(\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \varphi_{i+1}, \ldots \varphi_n)$
$= \exists y . (y \in \varphi_i \wedge x \in \sigma(\varphi_1, \ldots, \varphi_{i-1}, y, \varphi_{i+1}, \ldots \varphi_n))$

D. Defining special relations

Here we show how to define special relations using patterns.

1) Functions: We can state that a symbol $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ is to be interpreted as a function in all models as follows:

$$\exists y \,.\, \sigma(x_1, \ldots, x_n) = y$$

Indeed, if M is any model satisfying the pattern above and $a_1 \in M_{s_1}, \ldots, a_n \in M_{s_n}$ then let $\rho : Var \to M$ be an M-valuation such that $\rho(x_1) = a_1, \ldots, \rho(x_n) = a_n$. Since M satisfies the pattern, it follows that $M_s = \bigcup\{\overline{\rho'}(\sigma(x_1, \ldots, x_n) = y) \mid \rho' : Var \to M,\ \rho'\!\upharpoonright_{Var \setminus \{x\}} = \rho\!\upharpoonright_{Var \setminus \{x\}}\}$. Since $\overline{\rho'}(\sigma(x_1, \ldots, x_n) = y)$ is either M_s or $\emptyset$, depending upon whether $\sigma_M(x_1, \ldots, x_n) = \{\rho'(y)\}$ holds or not, we conclude that there exists some $\rho' : Var \to M$ such that $\sigma_M(a_1, \ldots, a_n) = \{\rho'(y)\}$, that is, $\sigma_M(a_1, \ldots, a_n)$ is a one-element set. Therefore, σ_M represents a total function.

To avoid writing such boring function patterns, from now on we automatically assume such an axiom whenever we write a symbol $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ using the function notation

$$\sigma : s_1 \times \cdots \times s_n \to s$$

2) Injective functions: $(f(x) = f(y)) \to (x = y)$ states that f is injective. If $(M, f_M : M \to M)$ is any model satisfying this specification, then f_M must be injective. Indeed, let $a, b \in M$ such that $a \neq b$ and $f_M(a) = f_M(b)$. Pick $\rho : Var \to M$ such that $\rho(x) = a$ and $\rho(y) = b$. Since M satisfies the axiom above, we get $\overline{\rho}(f(x) = f(y)) \subseteq \overline{\rho}(x = y)$. But Proposition 6 implies that $\overline{\rho}(x = y) = \emptyset$ and $\overline{\rho}(f(x) = f(y)) = M$, which is a contradiction. We can also show that any model whose f is injective satisfies the axiom. Let $(M, f_M : M \to M)$ be any model such that f_M is injective. It suffices to show $\overline{\rho}(f(x) = f(y)) \subseteq \overline{\rho}(x = y)$ for any $\rho : Var \to M$, which follows by Proposition 6: if $\rho(x) = \rho(y)$ then $\overline{\rho}(f(x) = f(y)) = \overline{\rho}(x = y) = M$, and if $\rho(x) \neq \rho(y)$ then $\overline{\rho}(f(x) = f(y)) = \overline{\rho}(x = y) = \emptyset$ because f_M is injective.

We write $\varphi \neq \varphi'$ instead of $\neg(\varphi = \varphi')$. With this, another way to capture the injectivity of f is $(x \neq y) \to (f(x) \neq f(y))$.

3) Partial functions: Partial functions $\sigma \in \Sigma_{s_1\ldots s_n,s}$ can be specified with $\sigma(x_1,\ldots,x_n) = \bot_s \vee \exists y\,.\,\sigma(x_1,\ldots,x_n) = y$ and from now on we use the notation (note the "$\rightharpoonup$" symbol)

$$\sigma : s_1 \times \cdots \times s_n \rightharpoonup s$$

to automatically assume a pattern like the above. For example, a division partial function which is undefined in all models when the denominator is 0 can be specified as follows:

$$\_/\_ : Nat \times Nat \rightharpoonup Nat \qquad\qquad \neg(x/0)$$

4) Total relations: Total relations can be defined with $\lceil\sigma(x_1,\ldots,x_n)\rceil_s^s$, equivalent to $\sigma(x_1,\ldots,x_n) \neq \bot_s$. We write

$$\sigma : s_1 \times \cdots \times s_n \Rightarrow s$$

to automatically assume that σ is a total relation.

E. Algebraic specifications and matching logic modulo theories

An algebraic specification consists of a many-sorted signature (S,Σ) together with a set of equations E over Σ-terms with variables. To translate an algebraic specification into a matching logic specification we only need to ensure that operation symbols get a function interpretation by adding axioms of the form described in the previous section, and to regard each equation $t = t'$ as an equality pattern $t = t'$.

Proposition 8: Let (S,Σ,F) be the matching logic specification associated to the algebraic specification (S,Σ,E) as above, that is, F interprets each equation in E as a pattern and adds patterns stating that the symbols in Σ are interpreted as functions. Then for any Σ-equation e, $E \models_{alg} e$ iff $F \models e$.

Using the notations introduced so far, the Peano natural numbers would be defined as follows in matching logic:

$$0 : \to Nat \qquad succ : Nat \to Nat \qquad plus : Nat \times Nat \to Nat$$
$$plus(0,y) = y \qquad\qquad plus(succ(x),y) = succ(plus(x,y))$$

This looks identical to the algebraic specification definition.

Note, however, that matching logic allows us to add more than just equational patterns. For example, we can add to F the pattern $0 \vee \exists x\,.\,succ(x)$ stating that any number is either 0 or the successor of another number. Nevertheless, since matching logic ultimately has the same expressive power as predicate logic (Proposition 2), we cannot finitely axiomatize in matching logic any mathematical domains which do not already admit finite FOL axiomatizations. In practice, we follow the same standard approach as the first-order SMT solvers, namely desired domains are theoretically presented with potentially infinitely many axioms but are implemented using specialized decision procedures. Indeed, our MatchC prover [11] defers to Z3 [4] the solving of all the domain constraints.

Algebraic specifications and decision procedures of mathematical domains abound in the literature. All of these can now be leveraged and used in the context of matching logic. We do not discuss these further, but only mention that from now on in this paper we tacitly assume definitions of integer and of natural numbers, as well as of Boolean values, with common operations on them. We assume that these come with three sorts, *Int*, *Nat* and *Bool*, and the operations on them use the conventional syntax and writing; e.g., $\_ \leq \_ : Nat \times Nat \to Bool$, $x \leq y$, etc. To compact writing, we take the freedom to write b instead of $b = true$ for Boolean expressions b, in any sort context. For example, we write $\varphi_s \wedge x \leq y$ instead of $\varphi_s \wedge (x \leq y =^s_{Bool} true)$.

F. First-Order Logic

First-order logic (FOL) allows both function and predicate symbols. The function symbols are used to build terms, and then predicates are defined over terms. Formally, the syntax of (many-sorted) FOL is defined as follows:

$$\begin{array}{lcll} t_s & ::= & x \in Var_s \mid \sigma(t_1,\ldots,t_n) & \text{where } \sigma \in \Sigma_{s_1\ldots s_n,s} \\ \varphi & ::= & \pi(x_1,\ldots,x_n) & \text{where } \pi \in \Pi_{s_1\ldots s_n} \\ & \mid & \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\varphi & \end{array}$$

Let (S,Σ,Π) be a FOL signature. Like in pure predicate logic, we add a *Pred* sort and regard the predicate symbols as symbols of result sort *Pred*. Specifically, let (S^{ML},Σ^{ML}) be the matching logic signature where $S^{ML} = S \cup \{Pred\}$ and where $\Sigma^{ML} = \Sigma \cup \{\pi : s_1 \ldots s_n \to Pred \mid \pi \in \Pi_{s_1\ldots s_n}\}$, and let F be the set

$$\{\exists z{:}s\,.\,\sigma(x_1{:}s_1,\ldots,x_n{:}s_n) = z \mid \sigma \in \Sigma_{s_1\ldots s_n,s}\}$$

requiring that each function symbol is interpreted as a function.

Proposition 9: For any FOL formula φ, $\models_{FOL} \varphi$ iff $F \models \varphi$.

IV. Maps, Separation Logic and Structural Framing

Matching logic has inherent support for separation, without a need for any special logic constructs or extensions. That is because pattern matching has a spatial meaning by its very nature, in that matching a subterm already separates that subterm from the rest of the context, so matching two or more terms can only happen when there is no overlapping between them. We show that separation logic reduces to matching logic reasoning within the canonical model of a straightforward specification of maps. The $P_1 \mathrel{-\!\!*} P_2$ construct of separation logic becomes an alias for the pattern $\exists H{:}Map\,.\,H \wedge \lceil H * P_1 \to P_2\rceil$.

A. Maps and Map Patterns

We start by specifying maps. We only discuss maps from natural numbers to natural numbers, but they can be similarly defined over arbitrary domains as keys and as values. We use a syntax for maps that resembles that of separation logic [8], but recall there are no predicates in what follows, only patterns.

$$\begin{array}{l} \_ \mapsto \_ : Nat \times Nat \rightharpoonup Map \\ emp : \to Map \\ \_ * \_ : Map \times Map \rightharpoonup Map \\[4pt] \neg(0 \mapsto a) \\ emp * H = H \\ H_1 * H_2 = H_2 * H_1 \\ (H_1 * H_2) * H_3 = H_1 * (H_2 * H_3) \\ (x \mapsto a * y \mapsto b) \to x \neq y \end{array}$$

Consider the canonical model of maps-as-heaps M, where: $M_{Nat} = \{0,1,2,\ldots\}$; M_{Map} = partial maps from natural numbers to natural numbers with finite domains and undefined in 0, with *emp* interpreted as the map undefined everywhere, with $\_ \mapsto \_$

interpreted as the corresponding one-element partial map except when the first argument is 0 in which case it is undefined (note that $\_ \mapsto \_$ was declared using $\rightharpoonup$), and with $\_ * \_$ interpreted as map merge when the two maps have disjoint domains, or undefined otherwise (note that $\_ * \_$ was also declared using $\rightharpoonup$). If $h, h' \in M_{Map}$ then we use the dedicated notation $h \# h'$ to denote the fact that maps h and h' are merge-able (their domains are disjoint). M satisfies all axioms above.

Let us now define a basic data-type over maps, the lists. We define two patterns, for complete lists and for list fragments:

$$\begin{array}{l} list : Nat \Rightarrow Map \\ list(0) = emp \\ list(x) \wedge x \neq 0 = \exists z\,.\, x \mapsto z * list(z) \end{array}$$

$$\begin{array}{l} lseg : Nat \times Nat \Rightarrow Map \\ lseg(x, x) = emp \\ lseg(x, y) \wedge x \neq y = \exists z\,.\, x \mapsto z * lseg(z, y) \end{array}$$

There are two questions: Does this specification admit any solution (i.e., interpretations $list_M : M_{Nat} \to \mathcal{P}(M_{Map})$ and $lseg_M : M_{Nat} \times M_{Nat} \to \mathcal{P}(M_{Map})$) in M? If yes, is the solution unique? We answer these positively. We only discuss $lseg_M$, because the other is similar and simpler.

A solution $lseg_M : M_{Nat} \times M_{Nat} \to \mathcal{P}(M_{Map})$ exists iff it satisfies the two axioms for $lseg$ above, that is,

$$\begin{array}{l} lseg_M(n, n) = \{emp_M\} \text{ for all } n \geq 0 \\ lseg_M(n, m) = \bigcup\{(\{n \mapsto_M n_1\} *_M lseg_M(n_1, m)) \mid n_1 \geq 0\}, n \neq m \end{array}$$

where $\_ *_M \_$ is M's merge function explained above extended to sets of maps for each argument; recall that the map merge function is undefined (i.e., it yields an empty set of maps) when the two argument maps are not merge-able.

First, we claim that the following is a solution:

$$\begin{array}{l} lseg_M(n, n) = \{emp_M\} \text{ for all } n \geq 0 \\ lseg_M(n, m) = \{\ n \mapsto_M n_1 *_M n_1 \mapsto_M n_2 *_M \cdots *_M n_k \mapsto_M m \\ \qquad\qquad \mid k \geq 0, \text{ and } n_0 = n, n_1, n_2, \ldots, n_k > 0 \text{ all different}\} \end{array}$$

Indeed, the first axiom vacuously holds, while for the second all we need to note is that the "junk" maps where n is 0 or in the domain of a map in $lseg_M(n_1, m)$ are simply discarded by the map merge interpretation of $\_ * \_$.

Second, the above is the unique solution. It suffices to prove, by induction on the size k of the domain of $h \in M_{Map}$ that: $h \in lseg_M(n, m)$ for $n, m \in M_{Nat}$ iff either $n = m$ and $h = emp_M$ (i.e., $k = 0$), or otherwise $n \neq 0$ and $n \neq m$ and $k > 0$ and there are distinct $n_0 = n$, n_1, …, n_k such that $h = (n \mapsto_M n_1 *_M n_1 \mapsto_M n_2 *_M \cdots *_M n_{k-1} \mapsto_M m)$. Since the maps in $lseg_M(n, m)$ when $n \neq 0$ and $n \neq m$ contain at least one binding, we conclude $k = 0$ can only happen iff $h \in lseg_M(n, n)$, and then $h = emp_M$. Now suppose $k > 0$, which can only happen iff $h \in lseg_M(n, m)$ for $n \neq m$, which can only happen iff $n \neq 0$ and $h = n \mapsto_M n_1 *_M h_1$ for some $n_1 \geq 0$ and $h_1 \in lseg_M(n_1, m)$. It all follows now from the induction hypothesis applied to h_1.

It should be clear that patterns can be specified many different ways. E.g., the first list pattern can also be specified as:

$$list(x) = (x = 0 \wedge emp \vee \exists z\,.\, x \mapsto z * list(z))$$

In a similar style, we can define more complex patterns, such as lists with data. But first, we specify a convenient operation for defining maps over contiguous locations/keys, making use of a sequence data-type; the latter can be defined, for example, using a conventional algebraic specification style (Section III-E) with an associative binary comma construct for sequences and a unit ϵ (see, e.g., Appendix I):

$$\begin{array}{l} \_ \mapsto [\_] : Nat \times Seq \to Map \\ x \mapsto [\epsilon] = emp \\ x \mapsto [a, S] = x \mapsto a * (x + 1) \mapsto [S] \end{array}$$

To continue the construction of our canonical model M, we take M_{Seq} to be the finite sequences of natural numbers, with $\epsilon : Seq$ and $\_ \cdot \_ : Seq \times Seq \to Seq$ interpreted as the empty sequence and, respectively, the sequence concatenation.

We can now define lists with data as follows:

$$\begin{array}{l} list : Nat \times Seq \Rightarrow Map \\ list(0, \epsilon) = emp \\ list(x, n \cdot S) = \exists z\,.\, x \mapsto [n, z] * list(z, S) \end{array}$$

$$\begin{array}{l} lseg : Nat \times Seq \times Nat \Rightarrow Map \\ lseg(x, \epsilon, x) = emp \\ lseg(x, n \cdot S, y) = \exists z\,.\, x \mapsto [n, z] * lseg(z, S, y) \end{array}$$

We can show, using a similar approach like for lists without data, that the pattern $lseg(x, S, y)$ matches in M precisely the lists starting with x, exiting to y, and holding data sequence S.

It is easy now to devise other similar data-type specifications:

$$\begin{array}{ll} none : \to Tree & node : Nat \times Tree \times Tree \to Tree \\ tree : Nat \times Tree \Rightarrow Map & \end{array}$$

$$\begin{array}{l} tree(0, none) = emp \\ tree(x, node(n, t_1, t_2)) = \exists y\, z\,.\, x \mapsto [n, y, z] * tree(y, t_1) * tree(z, t_2)) \end{array}$$

Therefore, fixing the interpretations of the basic mathematical domains, such as those of natural numbers, sequences, maps, etc., suffices in order to define interesting heap patterns, in the sense that the axioms themselves uniquely define the desired data-types. No "inductive predicates" or inductive principles of any kind were needed in this case (although they may be needed in other definitions). Note, however, that choosing the right basic mathematical domains is crucial. For example, if we allow the maps in M_{Map} to have infinite domains then the list patterns without data above (the first ones) also include infinite lists. The lists with data cannot include infinite lists, because we only allow finite sequences. This would, of course, change if we allow infinite sequences, or streams, in the model.

Although in the particular case of our basic domains chosen in M it turned out that $list$ and $lseg$ admit unique interpretations, that is neither general nor needed for the subsequent developments. For example, if we allow infinite-domain maps as described above, then $list$ and $lseg$ do not admit unique interpretations anymore, because we can interpret them to be either all the finite domain lists, or both the finite and the infinite-domain lists. That we can write patterns which admit the desired solution in the desired model suffices in practice, because our reasoning techniques developed in the rest of the paper allow us to derive properties that hold in all models satisfying the axioms, so any derived property is sound also for the intended model and interpretations.

B. Relationship to Separation Logic

We next formally capture the relationship between separation logic and matching logic. We only discuss the well-established separation logic variant in [8]. Moreover, here we only discuss separation logic as an assertion-language, used for specifying state properties, and not its extension as an axiomatic programming language semantic framework. We regard the latter as an orthogonal aspect, which can similarly be approached using matching logic. Assume some basic syntax for integer expressions coming with sorts *Int* and *Bool*, as discussed in Section III-E, which for simplicity we use both in the definition of separation logic and in its corresponding matching logic specification. Note that [8] uses a syntactic category name for integer expressions different from *Int*, and *Int* for the actual integer values in the model, but this is irrelevant. Then the syntax of separation logic (SL) can be defined as follows:

$$\begin{array}{rcll} Bool & ::= & \mathsf{isatom?}(Int) \mid \mathsf{isloc?}(Int) & \\ \varphi & ::= & Bool \mid Int \mapsto Int & \text{(Atomic)} \\ & \mid & \mathsf{false} \mid \varphi \to \varphi \mid \forall x.\varphi & \text{(Classic)} \\ & \mid & emp \mid \varphi * \varphi \mid \varphi \mathrel{-\!\!*} \varphi & \text{(Spatial)} \end{array}$$

The predicates isatom? and isloc? partition the set of integers into *atoms* and *locations*. The SL semantics is based on a model of *stores* and *heaps*, which are finite-domain maps from variables and respectively from locations to integers. The semantics of each syntactic construct is given in terms of a pair (s, h) of a store and a heap, called a *state*. For example, $(s, h) \models_{SL} E_1 \mapsto E_2$ iff $Dom(h) = \overline{s}(E_1)$ and $h(\overline{s}(E_1)) = \overline{s}(E_2)$, and $(s, h) \models_{SL} P_1 * P_2$ iff there exist $h_1 \# h_2$ such that $h = h_1 * h_2$ and $(s, h_1) \models_{SL} P_1$ and $(s, h_2) \models_{SL} P_2$. The semantics of $P_1 \mathrel{-\!\!*} P_2$ is the states whose heaps extended with a fragment satisfying P_1 result in ones satisfying P_2: $(s, h) \models_{SL} P_1 \mathrel{-\!\!*} P_2$ iff for any h_1 with $h \# h_1$, if $(s, h_1) \models_{SL} P_1$ then $(s, h * h_1) \models_{SL} P_2$.

Let us now define a matching logic specification and a model of it, which precisely capture the SL variant above. The sorts *Int* and *Bool*, as well as all the operation symbols on them are the same. The classic logic predicate constructs are already captured by the generic syntax of patterns. The heaplet and spatial constructs, except for the $\mathrel{-\!\!*}$, are given by the matching logic specification of maps discussed above, in which we substitute *Int* for *Nat*. The only additional specification is:

$$\begin{array}{l} \mathsf{isatom?},\ \mathsf{isloc?} : Int \to Bool \\ \_ \mathrel{-\!\!*} \_ : Map \times Map \rightharpoonup Map \end{array}$$

$$P_1 \mathrel{-\!\!*} P_2 = \exists H \,.\, H \wedge [H * P_1 \to P_2]$$

Recall from Section III-C that [_] is the symbol whose semantics leverages the non-emptyness of its argument to the total set. In words, $P_1 \mathrel{-\!\!*} P_2$ is the set of all maps h which merged with maps satisfying P_1 yield only maps satisfying P_2. Thanks to the matching logic notational convention that Booleans b stand for equalities $b = true$, a SL formula can be regarded, as is, as a matching logic pattern of sort *Map*.

We next construct our model. Let M be identical to the model for maps above, except that we replace natural numbers with integer numbers. We define $\mathsf{isatom?}_M : M_{Int} \to M_{Bool}$ and $\mathsf{isloc?}_M : M_{Int} \to M_{Bool}$ to partition the set of integers into atoms and locations the same way as in the SL semantics. The only thing left is to define the partial function $\_ \mathrel{-\!\!*}_M \_ : Map \times Map \to \mathcal{P}(Map)$, which we do as follows:

$$h_1 \mathrel{-\!\!*}_M h_2 = \{h \mid h \# h_1 \text{ and } h * h_1 = h_2\}$$

Note that $h_1 \mathrel{-\!\!*}_M h_2$ is either the empty set or it is a set of precisely one map. Then the following result holds:

Proposition 10: If φ is a SL formula, then $\models_{SL} \varphi$ iff $M \models \varphi$.

Proof: We show by structural induction on φ the more general result that for any store s and any heap h, we have $(s, h) \models \varphi$ iff $h \in \overline{s}(\varphi)$. The only interesting case is $\varphi \equiv \varphi_1 \mathrel{-\!\!*} \varphi_2$:

$$\begin{array}{ll} & h \in \overline{s}(\varphi_1 \mathrel{-\!\!*} \varphi_2) \\ \text{iff} & h \in \overline{s}(\exists H \,.\, H \wedge [H * \varphi_1 \to \varphi_2]) \\ \text{iff} & \{h\} *_M \overline{s}(\varphi_1) \subseteq \overline{s}(\varphi_2) \\ \text{iff} & h * h_1 \in \overline{s}(\varphi_2) \text{ for any } h_1 \in \overline{s}(\varphi_1) \text{ with } h \# h_1 \\ \text{iff} & (s, h * h_1) \models_{SL} \varphi_2 \text{ for any } h_1 \text{ with } (s, h_1) \models_{SL} \varphi_1 \text{ and } h \# h_1 \\ \text{iff} & (s, h) \models_{SL} \varphi_1 \mathrel{-\!\!*} \varphi_2 \qquad \blacksquare \end{array}$$

Discussion: The loose-model approach of matching logic is in sharp technical, but not conceptual, contrast to separation logic. In separation logic, the syntax of maps and separation constructs is part of the syntax of the logic itself, and the model of maps is intrinsically integrated within the semantics of the logic: its satisfaction relation is defined in terms of a fixed syntax and the fixed model of the basic domains (maps, sequences, etc.). Then specialized proof rules and theorem provers need to be devised. If any changes to the syntax or semantics are desired, for example adding a new stack, or an I/O buffer, etc., then a new logic is obtained. Proof rules and theorem provers may also need to change as the logic changes. In matching logic, the basic ingredients of separation logic become one particular theory, with a particular syntax and particular axioms, together with a particular but carefully chosen model. This enables us to use generic first-order reasoning with the axioms of matching logic theories (Section V), as well as theorem provers or SMT solvers for reasoning about the intended model (the M in Proposition 10).

C. Structural Framing

Heap framing is a major outcome of the use of separation logic for program verification, since it enables local reasoning:

$$\vdash \{\varphi_{pre}\}\, \mathsf{c}\, \{\varphi_{post}\} \text{ implies}^2 \vdash \{\varphi_{pre} * \varphi\}\, \mathsf{c}\, \{\varphi_{post} * \varphi\}$$

Matching logic can be identically used instead of separation logic in axiomatic semantics (see [13] for an early example). In particular, the heap framing rule above would stay unchanged.

However, if used in combination with reachability logic [11], [12], matching logic enables us to develop more flexible and more general types of framing. Regarding flexibility, note that we may not always want to automatically assume heap

[2]Depending on the language, the rule may also have side conditions on the locations accessed by c and φ, but those are irrelevant for our discussion here.

framing (e.g., when memory is finite—embedded systems, device drivers, etc.—, or when the language has functions like $\mathsf{getTotalMemory}()$ returning the available memory). Regarding generality, we may want similar framing rules for other semantic cells in the program configuration, such as input/output buffers, exception stacks, thread resources, etc.

Consider the (operational) semantic rule of assignment in a C-like language, whose configuration contains an environment map from variables to locations and a heap map from locations to values (say due to the "address" construct &):

$$\begin{array}{ll} & \langle\langle \mathsf{x} = \mathsf{v}; R\rangle_{\mathsf{k}}\ \langle \mathsf{x} \mapsto l * e\rangle_{\mathsf{env}}\ \langle l \mapsto \_ * h\rangle_{\mathsf{heap}}\ c\rangle_{\mathsf{cfg}} \\ \Rightarrow & \langle\langle \qquad R\rangle_{\mathsf{k}}\ \langle \mathsf{x} \mapsto l * e\rangle_{\mathsf{env}}\ \langle l \mapsto \mathsf{v} * h\rangle_{\mathsf{heap}}\ c\rangle_{\mathsf{cfg}} \end{array}$$

The variables R, e, h and c can all be thought of as *structural frames*: R is the code frame, e is the environment frame, h is the heap frame, and c is the configuration frame. The assignment rule above says that the value at the location l of x in the heap changes to v regardless of what the structural frames match.

The same specification style extends to arbitrary reachability properties, without a need to define an axiomatic semantics. For example, MatchC desugars the grayed rule specification of the function in Fig. 1 into the following reachability rule

$$\begin{array}{l} \langle\langle \mathsf{body}; R\rangle_{\mathsf{k}}\langle \mathsf{n} \mapsto l * e\rangle_{\mathsf{env}}\langle l \mapsto n * h\rangle_{\mathsf{heap}}\langle \mathsf{A}, I\rangle_{\mathsf{in}}\langle O\rangle_{\mathsf{out}}\ c\rangle_{\mathsf{cfg}} \\ \quad \wedge\ n = \mathsf{len}(\mathsf{A}) \\ \Rightarrow \exists n'.\langle\langle R\rangle_{\mathsf{k}}\langle \mathsf{n} \mapsto l * e\rangle_{\mathsf{env}}\langle l \mapsto n' * h\rangle_{\mathsf{heap}}\langle I\rangle_{\mathsf{in}}\langle O, \mathsf{rev}(\mathsf{A})\rangle_{\mathsf{out}}\ c\rangle_{\mathsf{cfg}} \end{array}$$

If we want to also state that n is not modified, then we remove the existential quantifier and replace n' with n. If we want to say, for whatever reason, that the heap must be empty when this function is invoked, then we remove the heap frame h. If we want to state that the size of the available memory must be larger than a certain limit, then we add the constraint $\mathsf{size}(h) \geq \mathit{limit}$ to the LHS pattern. Similarly for the other structural frames, O, I, and c. It should be clear that this gives us significant power in what kind of properties we can specify. Reachability logic [12] provides a language-independent sound and complete proof system to derive such reachability rules, starting with the formal (operational) semantics of the language.

If a language semantics is so that structural framing in a particular semantic cell is always sound, say in the $\langle ...\rangle_{\mathsf{heap}}$ cell, then one can prove a property "$\langle\langle h_1\rangle_{\mathsf{heap}}\ c_1\rangle_{\mathsf{cfg}} \Rightarrow \langle\langle h_2\rangle_{\mathsf{heap}}\ c_2\rangle_{\mathsf{cfg}}$ implies $\langle\langle h_1 * h\rangle_{\mathsf{heap}}\ c_1\rangle_{\mathsf{cfg}} \Rightarrow \langle\langle h_2 * h\rangle_{\mathsf{heap}}\ c_2\rangle_{\mathsf{cfg}}$ when side-condition". Such a property can be proved, e.g., when all semantic rules use an unconstrained heap frame h (like in the assignment rule above). However, such a rule would not hold in a language providing, e.g., a construct that returns the size of the available memory. It is worth mentioning that although possible, such structural framing rules are unnecessary. That is because the structural frames are plain first-order variables that obey the general pattern matching principles like the other variables, so nothing special needs to be done about them. Indeed, in MatchC, the only difference between a framed and an unframed variant of a property is the use of "...".

V. Sound and Complete Deduction

As shown in Section II-B, the proof system of predicate logic is sound for matching logic as is. Ideally, we would like the

FOL axioms and rules:

1. $\vdash$ propositional tautologies
2. Modus ponens: $\vdash \varphi_1$ and $\vdash \varphi_1 \to \varphi_2$ imply $\vdash \varphi_2$
3. $\vdash (\forall x\,.\,\varphi_1 \to \varphi_2) \to (\varphi_1 \to \forall x\,.\,\varphi_2)$ when $x \notin FV(\varphi_1)$
4. Universal generalization: $\vdash \varphi$ implies $\vdash \forall x\,.\,\varphi$
5. Substitution: $\vdash (\forall x\,.\,\varphi) \wedge (\exists y\,.\,\varphi' = y) \to \varphi[\varphi'/x]$
6. Equality introduction: $\vdash \varphi = \varphi$
7. Equality elimination: $\vdash \varphi_1 = \varphi_2 \wedge \varphi[\varphi_1/x] \to \varphi[\varphi_2/x]$

Membership axioms and rules:

8. $\vdash \forall x\,.\,x \in \varphi$ iff $\vdash \varphi$
9. $\vdash x \in y = (x = y)$ when $x, y \in \mathit{Var}$
10. $\vdash x \in \neg\varphi = \neg(x \in \varphi)$
11. $\vdash x \in \varphi_1 \wedge \varphi_2 = (x \in \varphi_1) \wedge (x \in \varphi_2)$
12. $\vdash (x \in \exists y.\varphi) = \exists y.(x \in \varphi)$, with x and y distinct
13. $\vdash x \in \sigma(\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \varphi_{i+1}, \ldots \varphi_n)$
$= \exists y.(y \in \varphi_i \wedge x \in \sigma(\varphi_1, \ldots, \varphi_{i-1}, y, \varphi_{i+1}, \ldots \varphi_n))$

Fig. 2. Sound and complete proof system of matching logic.

same to hold true for FOL with equality, that is, we would like its proof system to be sound as is for matching logic reasoning, where we replace terms and predicates with arbitrary patterns. This would enable us to use off-the-shelf FOL provers for matching logic reasoning with minimal changes.

Unfortunately, FOL's Substitution axiom, $(\forall x\,.\,\varphi) \to \varphi[t/x]$, is not sound if we replace t with any pattern. For example, consider the tautology $\forall x\,.\,\exists y\,.\,x = y$ and let φ be $\exists y\,.\,x = y$. If FOL's Substitution were sound for arbitrary patterns φ' instead of t, then the formula $\exists y\,.\,\varphi' = y$, stating that φ' evaluates to a unique element for any valuation, would be valid for any pattern φ'. However, this is not true in matching logic, because patterns can evaluate to any set of elements, including the empty set or the total set; several examples of such patterns were discussed in Section III. We need to modify Substitution to indicate that φ' admits unique evaluations:

Substitution: $\vdash (\forall x\,.\,\varphi) \wedge (\exists y\,.\,\varphi' = y) \to \varphi[\varphi'/x]$

Condition $\exists y\,.\,\varphi' = y$ holds when φ' is a term built with symbols σ obeying the functional axioms $\exists y\,.\,\sigma(x_1, \ldots, x_n) = y$ discussed in Section III-D. So the constrained substitution axiom is still more general that the original substitution axiom in FOL, since it can also apply when φ' is not built only from functional symbols but can be proved to have unique evaluation. It is interesting to note that a similar modification of Substitution was needed in the context of partial FOL [6], where the interpretations of functional symbols are partial functions, so terms may be undefined; axiom PFOL5 in [6] requires φ' to be *defined* in the Substitution rule, and several rules for proving definedness are provided. Note that our condition $\exists y\,.\,\varphi' = y$ is equivalent to definedness in the special case of PFOL, and that, thanks to the equality which can be defined in matching logic, we do not need special machinery for proving definedness.

Our approach to obtain a sound and complete proof system for matching logic is to build upon its reduction to predicate logic in Section II-B. Specifically, to use Proposition 2 and the complete proof system of predicate logic. Given a matching logic signature (S, Σ), let (S, Π_Σ) be the predicate logic signature obtained like in Section II-B. In addition to the PL translation there, we also define a backwards translation ML of (S, Π_Σ)-formulae into (S, Σ)-patterns inductively as follows:

$$\begin{array}{rcl}
ML(x = r) & = & x = r \\
ML(\pi_\sigma(r_1, \ldots, r_n, r)) & = & r \in \sigma(r_1, \ldots, r_n) \\
ML(\neg\psi) & = & \neg ML(\psi) \\
ML(\psi_1 \wedge \psi_2) & = & ML(\psi_1) \wedge ML(\psi_2) \\
ML(\exists x \,.\, \psi) & = & \exists x \,.\, ML(\psi) \\
ML(\{\psi_1, \ldots, \psi_n\}) & = & \{ML(\psi_1), \ldots, ML(\psi_n)\}
\end{array}$$

Recall from Section III-C that we tacitly assume equality and membership in all matching logic specifications.

Figure 2 shows our sound and complete proof system for matching logic reasoning, which was specifically crafted to include the proof system of first-order logic. Indeed, the first group of axiom and rule schemas include all the axioms and proof rules of FOL with equality as instances (the rules Substitution, Equation introduction and Equation elimination allow more general patterns instead of terms). The second group of proof rules, for reasoning about membership, is introduced for technical reasons, namely for the proof of Theorem 1. We have not used them so far in any of our program verification efforts using matching logic, and our current matching logic prover provides no reasoning support for membership.

Theorem 1: The proof system in Figure 2 is sound and complete for matching logic reasoning: $F \models \varphi$ iff $F \vdash \varphi$.

Proof: Propositions 3 and 7 showed the soundness of all rules except for Substitution. To show the soundness of Substitution, we show $\overline{\rho}((\forall x \,.\, \varphi) \wedge (\exists y \,.\, \varphi' = y)) \subseteq \overline{\rho}(\varphi[\varphi'/x])$ for any model M and valuation $\rho : Var \to M$. Let s be the sort of φ and s' be the sort of φ'. We have $\overline{\rho}((\forall x \,.\, \varphi) \wedge (\exists y \,.\, \varphi' = y)) = \bigcap\{\overline{\rho'}(\varphi) \mid \rho' \restriction_{Var\setminus\{x\}} = \rho \restriction_{Var\setminus\{x\}}\} \cap \bigcup\{M_s \mid \rho' \restriction_{Var\setminus\{x\}} = \rho \restriction_{Var\setminus\{x\}}, \overline{\rho'}(\varphi') = \{\rho'(y)\}\}$. Since $y \notin FV(\varphi')$, it follows that $\overline{\rho'}(\varphi') = \overline{\rho}(\varphi')$. Therefore, all we have to show is the following: if $\overline{\rho}(\varphi') = \{a\}$ for some $a \in M_{s'}$ then $\bigcap\{\overline{\rho'}(\varphi) \mid \rho' \restriction_{Var\setminus\{x\}} = \rho \restriction_{Var\setminus\{x\}}\} \subseteq \overline{\rho}(\varphi[\varphi'/x])$. This holds because $\overline{\rho}(\varphi[\varphi'/x]) = \overline{\rho[a/x]}(\varphi)$.

We now show the completeness. First, note that Proposition 2 and the completeness of predicate logic imply that $F \models \varphi$ iff $PL(F) \vdash^{=}_{PL} PL(\varphi)$. Second, note that $PL(F) \vdash^{=}_{PL} PL(\varphi)$ implies $ML(PL(F)) \vdash ML(PL(\varphi))$, because the ML translation only replaces predicates $\pi_\sigma(r_1, \ldots, r_n, r)$ with $r \in \sigma(r_1, \ldots, r_n)$ and the proof rules of predicate logic are a subset of the proof rules of matching logic. Third, notice that the completeness result holds if we can show $F \vdash \varphi$ iff $F \vdash ML(PL(\varphi))$ for any pattern φ: indeed, then $F \vdash ML(PL(F))$, which together with the above implies $F \vdash ML(PL(\varphi))$, which further implies $F \vdash \varphi$.

Let us now prove that $F \vdash \varphi$ iff $F \vdash ML(PL(\varphi))$ for any pattern φ. We first show $\vdash r \in \varphi = ML(PL(\varphi, r))$ by induction on φ. The cases $\varphi \equiv x$, $\varphi \equiv \neg\varphi'$, $\varphi \equiv \varphi_1 \wedge \varphi_2$, and $\varphi \equiv \exists y . \varphi'$ are immediate consequences of the axioms 9-12 in Figure 2, using the induction hypothesis and Equality elimination (rule 7). For the case $\varphi \equiv \sigma(\varphi_1, \ldots, \varphi_n)$, we can first derive $ML(PL(\varphi, r)) = \exists r_1 \cdots \exists r_n \,.\, r_1 \in \varphi_1 \wedge \cdots \wedge r_n \in \varphi_n \wedge r \in \sigma(r_1, \ldots, r_n)$ using the induction hypothesis and Equality elimination, and then $r \in \varphi = \exists r_1 \cdots \exists r_n \,.\, r_1 \in \varphi_1 \wedge \cdots \wedge r_n \in \varphi_n \wedge r \in \sigma(r_1, \ldots, r_n)$ using axiom 13 in Figure 2 and conventional FOL reasoning. Therefore, $\vdash r \in \varphi = ML(PL(\varphi, r))$. Our result now follows by rule 8 in Figure 2, since $ML(PL(\varphi)) \equiv \forall r \,.\, ML(PL(\varphi, r))$. ■

VI. Conclusion and Future Work

Matching logic is a sound and complete FOL variant that makes no distinction between function and predicate symbols. Its formulae, called patterns, mix symbols, logical connectives and quantifiers, and in models evaluate to sets of elements, those that "match" them, instead of just one value as terms do or a truth value as predicates do in FOL. Equality can be defined in matching logic, and several important variants of FOL fall as special fragments. Separation logic can be framed as a matching logic theory within the particular model of partial finite-domain maps, and heap patterns (lists, trees, etc.) elegantly specified using equations. Matching logic allows spatial specification and reasoning anywhere in a program configuration, and for any language, not only in the heap or other particular and fixed semantic components.

We made no efforts to minimize the number of rules in our proof system, because our main objective here was to include the proof system for FOL with equality. It is quite likely that an elegant minimal proof system working directly with the core symbols $[\_]_{s_1}^{s_2} \in \Sigma_{s_1,s_2}$ for all sorts $s_1, s_2 \in S$ can be obtained such that the equality and membership axioms and rules in Figure 2 can be proved as lemmas, but that was not our objective here. Likewise, we refrained from discussing any computationally effective fragments of matching logic in this paper, although MatchC implements such a fragment. Finally, complexity results in the style of [1], [2], [7] for separation logic can likely also be obtained for fragments of matching logic, both with respect to axioms and with respect to models.

References

[1] T. Antonopoulos, N. Gorogiannis, C. Haase, M. Kanovich, and J. Ouaknine. Foundations for decision problems in separation logic with general inductive predicates. In *FOSSACS'14*, LNCS. Springer, 2014. To appear.

[2] J. Brotherston, C. Fuhs, N. Gorogiannis, and J. Navarro Pérez. A decision procedure for satisfiability in separation logic with inductive predicates. Technical Report RN/13/15, University College London, 2013.

[3] M. Clavel, F. Durán, S. Eker, J. Meseguer, P. Lincoln, N. Martí-Oliet, and C. Talcott. *All About Maude*, volume 4350 of *LNCS*. Springer, 2007.

[4] L. De Moura and N. Bjørner. Z3: an efficient SMT solver. In *TACAS*, pages 337–340, 2008. LNCS 4963.

[5] C. Ellison and G. Roşu. An executable formal semantics of C with applications. In *POPL*, pages 533–544. ACM, 2012.

[6] W. M. Farmer and J. D. Guttman. A set theory with support for partial functions. *Studia Logica*, 66(1):59–78, 2000.

[7] R. Iosif, A. Rogalewicz, and J. Simácek. The tree width of separation logic with recursive definitions. In *CADE'13*, LNCS 7898, 2013.

[8] P. O'Hearn, J. Reynolds, and H. Yang. Local reasoning about programs that alter data structures. In *CSL*, pages 1–19. LNCS 2142, 2001.

[9] P. W. O'Hearn and D. J. Pym. The logic of bunched implications. *Bulletin of Symbolic Logic*, 5(2):215–244, 1999.

[10] J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In *LICS*, pages 55–74. IEEE, 2002.

[11] G. Roşu and A. Ştefănescu. Checking reachability using matching logic. In *OOPSLA*, pages 555–574. ACM, 2012.

[12] G. Roşu, A. Ştefănescu, Ştefan Ciobâcă, and B. M. Moore. One-path reachability logic. In *LICS*, pages 358–367. IEEE, 2013.

[13] G. Roşu, C. Ellison, and W. Schulte. Matching logic: An alternative to Hoare/Floyd logic. In *AMAST*, pages 142–162. LNCS 6486, 2010.

[14] A. Tarski and S. Givant. *A Formalization of Set Theory Without Variables*. Number 41. AMS, 1987.

Appendix

A. Proof of Proposition 1

- If $\rho_1, \rho_2 : Var \to M$, $\rho_1{\restriction_{FV(\varphi)}} = \rho_2{\restriction_{FV(\varphi)}}$ then $\overline{\rho_1}(\varphi) = \overline{\rho_2}(\varphi)$

Structural induction on φ. The only interesting case is when φ has the form $\exists x.\varphi'$, so $FV(\varphi) = FV(\varphi') \setminus \{x\}$. Then

$$\begin{aligned}
\overline{\rho_1}(\exists x.\varphi') &= \bigcup\{\overline{\rho_1'}(\varphi') \mid \rho_1' : Var \to M,\ \rho_1'{\restriction_{Var\setminus\{x\}}} = \rho_1{\restriction_{Var\setminus\{x\}}}\} \\
&\quad \text{(by the induction hypothesis)} \\
&= \bigcup\{\overline{\rho_1'}(\varphi') \mid \rho_1' : Var \to M,\ \rho_1'{\restriction_{FV(\varphi)}} = \rho_1{\restriction_{FV(\varphi)}}\} \\
&\quad \text{(since } \rho_1{\restriction_{FV(\varphi)}} = \rho_2{\restriction_{FV(\varphi)}}\text{)} \\
&= \bigcup\{\overline{\rho_2'}(\varphi') \mid \rho_2' : Var \to M,\ \rho_2'{\restriction_{FV(\varphi)}} = \rho_2{\restriction_{FV(\varphi)}}\} \\
&\quad \text{(by the induction hypothesis)} \\
&= \bigcup\{\overline{\rho_2'}(\varphi') \mid \rho_2' : Var \to M,\ \rho_2'{\restriction_{Var\setminus\{x\}}} = \rho_2{\restriction_{Var\setminus\{x\}}}\} \\
&= \overline{\rho_2}(\exists x.\varphi')
\end{aligned}$$

- If $x \in Var_s$ then $M \models x$ iff $|M_s| = 1$

$M \models x$ iff $\overline{\rho}(x) = M_s$ for all $\rho : Var \to M$, iff $\{\rho(x)\} = M_s$ for all $\rho : Var \to M$, iff M_s has only one element.

- If $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ and $\varphi_1, \ldots, \varphi_n$ are patterns of sorts $s_1, \ldots, s_n$, respectively, then $M \models \sigma(\varphi_1, \ldots, \varphi_n)$ iff $\sigma_M(\overline{\rho}(\varphi_1), \ldots \overline{\rho}(\varphi_n)) = M_s$ for any $\rho : Var \to M$

$M \models \sigma(\varphi_1, \ldots, \varphi_n)$ iff $\overline{\rho}(\sigma(\varphi_1, \ldots, \varphi_n)) = M_s$ for all $\rho : Var \to M$, iff $\sigma_M(\overline{\rho}(\varphi_1), \ldots \overline{\rho}(\varphi_n)) = M_s$ for any $\rho : Var \to M$.

- $M \models \neg\varphi$ iff $\overline{\rho}(\varphi) = \emptyset$ for any $\rho : Var \to M$

$M \models \neg\varphi$ iff $\overline{\rho}(\neg\varphi) = M_s$ for any $\rho : Var \to M$, iff $M_s \setminus \overline{\rho}(\varphi) = M_s$ for any $\rho : Var \to M$, iff $\overline{\rho}(\varphi) = \emptyset$ for any $\rho : Var \to M$.

- $M \models \varphi_1 \wedge \varphi_2$ iff $M \models \varphi_1$ and $M \models \varphi_2$

$M \models \varphi_1 \wedge \varphi_2$ iff $\overline{\rho}(\varphi_1 \wedge \varphi_2) = M_s$ for any $\rho : Var \to M$, iff $\overline{\rho}(\varphi_1) \cap \overline{\rho}(\varphi_2) = M_s$ for any $\rho : Var \to M$, iff $\overline{\rho}(\varphi_1) = M_s$ and $\overline{\rho}(\varphi_2) = M_s$ for any $\rho : Var \to M$, iff $M \models \varphi_1$ and $M \models \varphi_2$.

- If $\exists x.\varphi_s$ is closed, then $M \models \exists x.\varphi_s$ iff $\bigcup\{\overline{\rho}(\varphi_s) \mid \rho : Var \to M\} = M_s$; in particular, $M \models \exists x.x$

$M \models \exists x.\varphi_s$ iff $\overline{\rho}(\exists x.\varphi_s) = M_s$ for any $\rho : Var \to M$, iff $\bigcup\{\overline{\rho'}(\varphi_s) \mid \rho' : Var \to M,\ \rho'{\restriction_{Var\setminus\{x\}}} = \rho{\restriction_{Var\setminus\{x\}}}\} = M_s$ for any $\rho : Var \to M$, iff (by the first property in this proposition, since $FV(\varphi_s) \subseteq \{x\}$) $\bigcup\{\overline{\rho'}(\varphi_s) \mid \rho' : Var \to M\} = M_s$ for any $\rho : Var \to M$, iff $\bigcup\{\overline{\rho}(\varphi_s) \mid \rho : Var \to M\} = M_s$. In particular, if $\varphi_s = x$ then $\bigcup\{\overline{\rho}(x) \mid \rho : Var \to M\} = M_s$, so $M \models \exists x.x$.

- $M \models \varphi_1 \to \varphi_2$ iff $\overline{\rho}(\varphi_1) \subseteq \overline{\rho}(\varphi_2)$ for all $\rho : Var \to M$

$M \models \varphi_1 \to \varphi_2$ iff $\overline{\rho}(\varphi_1 \to \varphi_2) = M$ for all $\rho : Var \to M$, iff $\overline{\rho}(\neg(\varphi_1 \wedge \neg\varphi_2)) = M$ for all $\rho : Var \to M$, iff $\overline{\rho}(\varphi_1 \wedge \neg\varphi_2) = \emptyset$ for all $\rho : Var \to M$, iff $\overline{\rho}(\varphi_1) \cap (M \setminus \overline{\rho}(\varphi_2)) = \emptyset$ for all $\rho : Var \to M$, iff $\overline{\rho}(\varphi_1) \subseteq \overline{\rho}(\varphi_2)$ for all $\rho : Var \to M$.

- $M \models \varphi_1 \leftrightarrow \varphi_2$ iff $\overline{\rho}(\varphi_1) = \overline{\rho}(\varphi_2)$ for all $\rho : Var \to M$

Follows from the previous similar properties for $\wedge$ and $\to$.

- $M \models \forall x.\varphi$ iff $M \models \varphi$

$M \models \forall x.\varphi$ iff $\overline{\rho}(\forall x.\varphi) = \bigcap\{\overline{\rho'}(\varphi) \mid \rho' : Var \to M,\ \rho'{\restriction_{Var\setminus\{x\}}} = \rho{\restriction_{Var\setminus\{x\}}}\} = M$ for all $\rho : Var \to M$, iff $\overline{\rho'}(\varphi) = M$ for all $\rho, \rho' : Var \to M$ with $\rho'{\restriction_{Var\setminus\{x\}}} = \rho{\restriction_{Var\setminus\{x\}}}$, iff $\overline{\rho}(\varphi) = M$ for all $\rho : Var \to M$, iff $M \models \varphi$.

B. Proof of Proposition 3

Propositional tautologies

They hold because $(\mathcal{P}(M), \neg_M, \cap)$ is a model of propositional logic for any set M. A similar argument is used underneath Proposition 4.

Modus ponens: $\models \varphi_1$ and $\models \varphi_1 \to \varphi_2$ imply $\models \varphi_2$

If $\rho : Var \to M$ is a matching logic model valuation such that $\overline{\rho}(\varphi_1) = M_{Pred}$ and $\overline{\rho}(\varphi_1) \subseteq \overline{\rho}(\varphi_2)$, then $\overline{\rho}(\varphi_2) = M_{Pred}$.

$\models (\forall x\,.\,\varphi_1 \to \varphi_2) \to (\varphi_1 \to \forall x\,.\,\varphi_2)$ when $x \notin FV(\varphi_1)$

Follows by properties in Proposition 1.

Universal generalization: $\models \varphi$ implies $\models \forall x\,.\,\varphi$

Immediate by Proposition 1.

C. Proof of Proposition 4

The implication "$\models_{Prop} \varphi$ implies $\models \varphi$" follows by 1) in Proposition 3. The other implication follows by the 1) and 2) in Proposition 3, noting that propositional logic admits complete deduction using a subset of propositional tautologies as axioms and modus ponens.

D. Proof of Proposition 5

Any predicate logic model $(\{M_s\}_{s\in S}, \{\pi_M\}_{\pi\in\Pi})$ extends into a matching logic model $(\{M_s\}_{s\in S\cup\{Pred\}}, \{\pi_M\}_{\pi\in\Sigma})$, where $M_{Pred} = \{1\}$ and $\pi_M(a_1, \ldots, a_n) = \{1\}$ iff $\pi_M(a_1, \ldots, a_n)$ holds, and $\pi_M(a_1, \ldots, a_n) = \emptyset$ otherwise. Therefore, $\models \varphi$ implies $\models_{PL} \varphi$. The converse follows by Proposition 3, which shows that each of the four proof rules of the complete proof system of pure predicate logic is sound for matching logic.

E. Proof of Proposition 6

1) $\overline{\rho}(\varphi =_{s_1}^{s_2} \varphi') = \emptyset$ iff $\overline{\rho}(\varphi) \neq \overline{\rho}(\varphi')$

Since $\varphi =_{s_1}^{s_2} \varphi' = \neg[\neg(\varphi \leftrightarrow \varphi')]_{s_1}^{s_2}$, we have $\overline{\rho}(\varphi =_{s_1}^{s_2} \varphi')$ equal to $M_{s_2} \setminus ([\_]_{s_1}^{s_2})_M(M_{s_1} \setminus (M_{s_1} \setminus (\overline{\rho}(\varphi_1)\ \Delta\ \overline{\rho}(\varphi_2))))$, which is further equal to $M_{s_2} \setminus ([\_]_{s_1}^{s_2})_M(\overline{\rho}(\varphi_1)\ \Delta\ \overline{\rho}(\varphi_2))$. So $\overline{\rho}(\varphi =_{s_1}^{s_2} \varphi') = \emptyset$ iff $([\_]_{s_1}^{s_2})_M(\overline{\rho}(\varphi_1)\ \Delta\ \overline{\rho}(\varphi_2)) = M_{s_2}$, iff $\overline{\rho}(\varphi_1)\ \Delta\ \overline{\rho}(\varphi_2) \neq \emptyset$, iff $\overline{\rho}(\varphi) \neq \overline{\rho}(\varphi')$.

2) $\overline{\rho}(\varphi =_{s_1}^{s_2} \varphi') = M_{s_2}$ iff $\overline{\rho}(\varphi) = \overline{\rho}(\varphi')$

Similarly to the property above, we have $\overline{\rho}(\varphi =_{s_1}^{s_2} \varphi') = M_{s_2}$ iff $([\_]_{s_1}^{s_2})_M(\overline{\rho}(\varphi_1)\ \Delta\ \overline{\rho}(\varphi_2)) = \emptyset$, iff $\overline{\rho}(\varphi_1)\ \Delta\ \overline{\rho}(\varphi_2) = \emptyset$, iff $\overline{\rho}(\varphi) = \overline{\rho}(\varphi')$.

3) $\models \varphi =_{s_1}^{s_2} \varphi'$ iff $\models \varphi \leftrightarrow \varphi'$

$\models \varphi =_{s_1}^{s_2} \varphi'$ iff $M \models \varphi =_{s_1}^{s_2} \varphi'$ for any model M, iff $\overline{\rho}(\varphi =_{s_1}^{s_2} \varphi') = M_{s_2}$ for any model M and $\rho : Var \to M$, iff $\overline{\rho}(\varphi) = \overline{\rho}(\varphi')$ for any model M and $\rho : Var \to M$, iff (by Proposition 1) $M \models \varphi \leftrightarrow \varphi'$ for any model M, iff $\models \varphi \leftrightarrow \varphi'$.

4) $\overline{\rho}(x \in_{s_1}^{s_2} \varphi) = \emptyset$ iff $\rho(x) \notin \overline{\rho}(\varphi)$

Since $x \in_{s_1}^{s_2} \varphi$ is equivalent to $[x \wedge \varphi]_{s_1}^{s_2}$, we have $\overline{\rho}(x \in_{s_1}^{s_2} \varphi) = ([\_]_{s_1}^{s_2})_M(\{\rho(x)\} \cap \overline{\rho}(\varphi))$, so $\overline{\rho}(x \in_{s_1}^{s_2} \varphi) = \emptyset$ iff $\{\rho(x)\} \cap \overline{\rho}(\varphi) = \emptyset$, that is, iff $\rho(x) \notin \overline{\rho}(\varphi)$.

5) $\overline{\rho}(x \in_{s_1}^{s_2} \varphi) = M_{s_2}$ iff $\rho(x) \in \overline{\rho}(\varphi)$

Similarly to above, $\overline{\rho}(x \in_{s_1}^{s_2} \varphi) = M_{s_2}$ iff $\{\rho(x)\} \cap \overline{\rho}(\varphi) \neq \emptyset$, that is, iff $\rho(x) \in \overline{\rho}(\varphi)$.

6) $\models x \in_{s_1}^{s_2} \varphi =_{s_2}^{s_3} (x \wedge \varphi =_{s_1}^{s_2} x)$

Let M be some model and $\rho : Var \to M$. By 1) and 2) above, the property holds iff we can show $\overline{\rho}(x \in_{s_1}^{s_2} \varphi) = \overline{\rho}(x \wedge \varphi =_{s_1}^{s_2} x)$. Since the membership and equality patterns evaluate either to the entire set or to the empty set, the following completes the proof: by 5) we have $\overline{\rho}(x \in_{s_1}^{s_2} \varphi) = M_{s_2}$ iff $\rho(x) \in \overline{\rho}(\varphi)$, iff $\{\rho(x)\} \cap \overline{\rho}(\varphi) = \{\rho(x)\}$, iff, by 2), $\overline{\rho}(x \wedge \varphi =_{s_1}^{s_2} x) = M_{s_2}$.

F. Proof of Proposition 7

1) Equality introduction: $\models \varphi = \varphi$

By 3) in Proposition 6

2) Equality elimination: $\models \varphi_1 = \varphi_2 \wedge \varphi[\varphi_1/x] \to \varphi[\varphi_2/x]$

Let M be some model and $\rho : Var \to M$. By Proposition 1, it suffices to show $\overline{\rho}(\varphi_1 = \varphi_2) \cap \overline{\rho}(\varphi[\varphi_1/x]) \subseteq \overline{\rho}(\varphi[\varphi_2/x])$. If $\overline{\rho}(\varphi_1) \neq \overline{\rho}(\varphi_1)$ then $\overline{\rho}(\varphi_1 = \varphi_2) = \emptyset$ by Proposition 6, so the inclusion holds. Now suppose that $\overline{\rho}(\varphi_1) = \overline{\rho}(\varphi_1)$, which implies $\overline{\rho}(\varphi_1 = \varphi_2) = M$ by Proposition 6, so it suffices to show $\overline{\rho}(\varphi[\varphi_1/x]) \subseteq \overline{\rho}(\varphi[\varphi_2/x])$. The stronger result $\overline{\rho}(\varphi[\varphi_1/x]) = \overline{\rho}(\varphi[\varphi_2/x])$ in fact holds, because the first element is a function of $\overline{\rho}(\varphi_1)$, the second element is the same function but of $\overline{\rho}(\varphi_2)$, and $\overline{\rho}(\varphi_1) = \overline{\rho}(\varphi_2)$.

3) $\models \forall x . x \in \varphi$ iff $\models \varphi$

Let M be a model. Then $M \models \forall x . x \in \varphi$ iff $M \models x \in \varphi$ (Proposition 1), iff $\overline{\rho}(x \in \varphi) = M$ for any $\rho : Var \to M$, iff $\rho(x) \in \overline{\rho}(\varphi)$ for any $\rho : Var \to M$ (by Proposition 6), iff $\overline{\rho}(\varphi) = M$ for any $\rho : Var \to M$, iff $M \models \varphi$.

4) $\models x \in y = (x = y)$ when $x, y \in Var$

By Proposition 6, it suffices to show $\overline{\rho}(x \in y) = M$ iff $\overline{\rho}(x = y) = M$ for any model M and any $\rho : Var \to M$, that is, also by Proposition 6, that $\rho(x) \in \{\rho(y)\}$ iff $\rho(x) = \rho(y)$, which obviously holds.

5) $\models x \in \neg\varphi = \neg(x \in \varphi)$

By Proposition 6, it suffices to show $\overline{\rho}(x \in \neg\varphi) = M$ iff $\overline{\rho}(x \in \varphi) = \emptyset$ for any model M and any $\rho : Var \to M$, that is, also by Proposition 6, that $\rho(x) \in M \backslash \overline{\rho}(\varphi)$ iff $\rho(x) \notin \overline{\rho}(\varphi)$, which obviously holds.

6) $\models x \in \varphi_1 \wedge \varphi_2 = (x \in \varphi_1) \wedge (x \in \varphi_2)$

By Proposition 6, it suffices to show $\rho(x) \in \overline{\rho}(\varphi_1) \cap \overline{\rho}(\varphi_2)$ iff $\rho(x) \in \overline{\rho}(\varphi_1)$ and $\rho(x) \in \overline{\rho}(\varphi_2)$ for any model M and any $\rho : Var \to M$, which obviously holds.

7) $\models x \in \exists y.\varphi = \exists y.(x \in \varphi)$, with x and y distinct

By Proposition 6, it suffices to show for any model M and any $\rho : Var \to M$, that $\rho(x) \in \bigcup\{\overline{\rho'}(\varphi) \mid \rho' : Var \to M,\ \rho' \restriction_{Var\backslash\{y\}} = \rho \restriction_{Var\backslash\{y\}}\}$ iff $\bigcup\{\overline{\rho'}(x \in \varphi) \mid \rho' : Var \to M,\ \rho' \restriction_{Var\backslash\{y\}} = \rho \restriction_{Var\backslash\{y\}}\} = M$. It is easy to see that each of the two statements holds iff there exists some $\rho' : Var \to M$ with $\rho' \restriction_{Var\backslash\{y\}} = \rho \restriction_{Var\backslash\{y\}}$ such that $\rho(x) \in \overline{\rho'}(\varphi)$.

8) $\models x \in \sigma(\varphi_1, \ldots, \varphi_{i-1}, \varphi_i, \varphi_{i+1}, \ldots, \varphi_n)$
$= \exists y.(y \in \varphi_i \wedge x \in \sigma(\varphi_1, \ldots, \varphi_{i-1}, y, \varphi_{i+1}, \ldots \varphi_n))$

By Proposition 6, it suffices to prove for any model M and any valuation $\rho : Var \to M$, that

$$\rho(x) \in \sigma_M(\overline{\rho}(\varphi_1), \ldots, \overline{\rho}(\varphi_{i-1}), \overline{\rho}(\varphi_i), \overline{\rho}(\varphi_{i+1}), \ldots, \overline{\rho}(\varphi_n))$$

iff there exists some $\rho' : Var \to M$ with $\rho' \restriction_{Var\backslash\{y\}} = \rho \restriction_{Var\backslash\{y\}}$ such that $\rho'(y) \in \overline{\rho}(\varphi_i)$ and

$$\rho(x) \in \sigma_M(\overline{\rho}(\varphi_1), \ldots, \overline{\rho}(\varphi_{i-1}), \{\rho'(y)\}, \overline{\rho}(\varphi_{i+1}), \ldots, \overline{\rho}(\varphi_n))$$

which obviously holds.

G. Proof of Proposition 8

The key observation is that, in a similar style to the proof of Proposition 5, there is a bijection between the matching logic models M satisfying F and the (S, Σ)-algebras M' satisfying E, such that $M \models e$ iff $M' \models_{alg} e$ for any Σ-equation e. The model bijection is defined as follows:

- $M'_s = M_s$ for each sort $s \in S$;
- $\sigma_{M'} : M_{s_1} \times \cdots \times M_{s_n} \times M_s$ with $\sigma_{M'}(a_1, \ldots, a_n) = a$ iff $\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$ with $\sigma_M(a_1, \ldots, a_n) = \{a\}$.

H. Proof of Proposition 9

The proof is similar to that of Proposition 5. Like there, any FOL model extends into a matching logic model whose *Pred* carrier contains only one element. Moreover, that model satisfies F because the interpretations of all the function symbols are functions. Therefore, $F \models \varphi$ implies $\models_{FOL} \varphi$.

Conversely, we can show exactly as in the proof of Proposition 5 that the axioms and proof rules of pure predicate logic are also sound here. FOL has only one additional proof rule:

Substitution: $\vdash (\forall x{:}s . \varphi) \to \varphi[t/x]$, with t of sort s

Let $M \models F$ be any matching logic model and let $\rho : Var \to M$. Then $\overline{\rho}(\forall x . \varphi) = \bigcap\{\overline{\rho'}(\varphi) \mid \rho' \restriction_{Var\backslash\{x\}} = \rho \restriction_{Var\backslash\{x\}}\} \subseteq \overline{\rho'}(\varphi)$ where $\rho' \restriction_{Var\backslash\{x\}} = \rho \restriction_{Var\backslash\{x\}}$ and $\{\rho'(x)\} = \overline{\rho}(t)$. Such ρ' can only be $\rho' = \rho[m/x]$ where $\overline{\rho}(t) = \{m\}$, so $\overline{\rho'}(\varphi) = \overline{\rho}(\varphi[t/x])$.

I. Sequences, Multisets and Sets

Sequences, multisets and sets are typical data-types defined using algebraic specification. Matching logic enables, however, some useful developments and shortcuts. For simplicity, we only discuss collections over *Nat*, and name the corresponding sorts *Seq*, *MultiSet*, and *Set*. Ideally, we would build upon an order-sorted algebraic signature setting, so that we can regard $x{:}Nat$ not only as an element of sort *Nat*, but also as one of sort *Seq* (a one-element sequence), as one of sort *MultiSet*, as well as one of sort *Set*. Extending matching logic to an order-sorted setting is not difficult, but would deviate from our main objective in this paper, so we refrain from doing it. Instead, we rely on the reader to assume either that order-sortedness does not bring additional complications (besides those of order-sortedness itself in the context of algebraic specification) or that elements of sort *Nat* used in a *Seq*, *MultiSet*, or *Set* context are wrapped with some injection symbol; e.g., $Nat2Seq(x)$.

Sequences (assume $Nat < Seq$, as discussed above) can be defined with two symbols and corresponding equations:

$$\epsilon : \to Seq \qquad \_\cdot\_ : Seq \times Seq \to Seq$$

$$\epsilon \cdot x = x \qquad x \cdot \epsilon = x \qquad (x \cdot y) \cdot z = x \cdot (y \cdot z)$$

We assume that lower cap variables have sort *Nat*, and upper cap ones have the appropriate collection sort. To avoid adding initiality constraints on models yet be able to do proofs by case analysis and elementwise equality, we may add the following:

$$\epsilon \vee \exists x\,.\,\exists S\,.\,x \cdot S \qquad\qquad (x \cdot S = x' \cdot S') = (x = x') \wedge (S = S')$$

We next define some operations on sequences:

$$\begin{array}{lll}
rev : Seq \rightarrow Seq & \_ \in \_ : Nat \times Seq \rightarrow Bool & \\
rev(\epsilon) = \epsilon & \neg(x \in \epsilon) & x \in x \cdot S \\
rev(x \cdot S) = rev(S) \cdot x & x \in y \cdot S \wedge (x \neq y) = x \in S &
\end{array}$$

We can transform sequences into multisets adding the equality axiom $x \cdot y = y \cdot x$, and into sets adding also the idemoptence equality $x \cdot x = x$. Here are some set operations:

$$\_ \cap \_ : Set \times Set \rightarrow Set \qquad\qquad \_\Delta\_ : Set \times Set \rightarrow Set$$

$$\begin{array}{l}
\epsilon \cap S_2 = S_2 \\
x \cdot S_1 \cap S_2 = ((x \in S_2 \rightarrow x) \wedge (\neg(x \in S_2) \rightarrow \epsilon)) \cdot (S_1 \cap S_2) \\
x \cdot S_1 \; \Delta \; x \cdot S_2 = S_1 \; \Delta \; S_2 \\
(S_1 \cap S_2 = \epsilon) \rightarrow (S_1 \; \Delta \; S_2 = S_1 \cdot S_2)
\end{array}$$