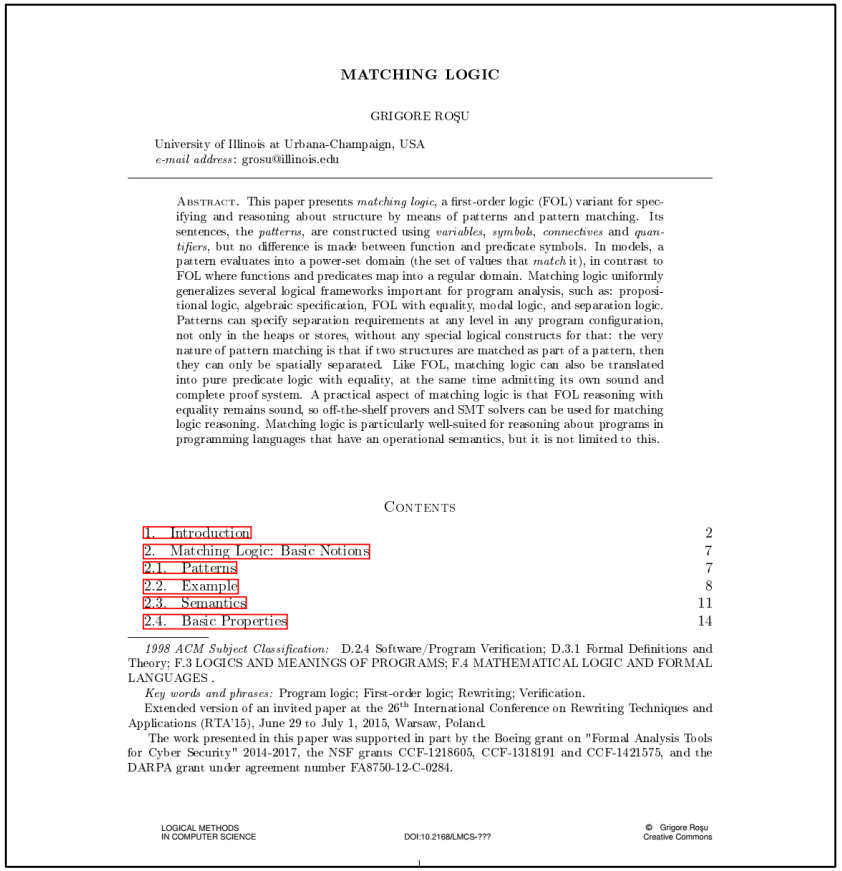
Matching Logic

Review of the paper "*Matching Logic*" by Grigore Rosu (published in *Logical Methods in Computer Science*, v. 13, pp. 1-61, 2017, available at https://fsl.cs.illinois.edu/publications/rosu-2017-lmcs.html)

Presenter: Nikolay V. Shilov

Review outline (and timeline)

- October 21, 2022 – *Basic Notions: Example, Syntax, Semantics, Basic Properties*
- October 28, 2022 – *Instances: Propositional Calculus, (Pure) Predicate Logic, First-Order Logic, Modal Logic, Separation Logic*
- November 1, 2022 – *Reduction to Predicate Logic with Equality, Sound and Complete Deduction*



MATCHING LOGIC

GRIGORE ROŞU

University of Illinois at Urbana-Champaign, USA
e-mail address: grosu@illinois.edu

Abstract. This paper presents *matching logic*, a first-order logic (FOL) variant for specifying and reasoning about structure by means of patterns and pattern matching. Its sentences, the *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 and predicates map into a regular domain. Matching logic uniformly generalizes several logical frameworks important for program analysis, such as: propositional logic, algebraic specification, FOL with equality, modal logic, and separation logic. Patterns can specify 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, at the same time admitting its own sound and complete proof system. A practical aspect of matching logic is that FOL reasoning with equality remains sound, so off-the-shelf provers and SMT solvers can be used for matching logic reasoning. Matching logic is particularly well-suited for reasoning about programs in programming languages that have an operational semantics, but it is not limited to this.

Contents

1998 ACM Subject Classification: D.2.4 Software/Program Verification; D.3.1 Formal Definitions and Theory; F.3 LOGICS AND MEANINGS OF PROGRAMS; F.4 MATHEMATICAL LOGIC AND FORMAL LANGUAGES .

Key words and phrases: Program logic; First-order logic; Rewriting; Verification.

Extended version of an invited paper at the 26th International Conference on Rewriting Techniques and Applications (RTA'15), June 29 to July 1, 2015, Warsaw, Poland.

The work presented in this paper was supported in part by the Boeing grant on "Formal Analysis Tools for Cyber Security" 2014-2017, the NSF grants CCF-1218605, CCF-1318191 and CCF-1421575, and the DARPA grant under agreement number FA8750-12-C-0284.

LOGICAL METHODS IN COMPUTER SCIENCE DOI:10.2168/LMCS-??? © Grigore Roşu Creative Commons

Basic Notation

October 21, 2022

Motivating and illustrative example

Consider the operational semantics of a real language like C, whose configuration has more than 100 semantic components [ref. below] . The semantic components, here called "cells" and written using symbols $\langle ... \rangle_{\mathsf{cell}}$, can be nested and their grouping (symbol) is governed by associativity and commutativity axioms. There is a top cell $\langle ... \rangle_{\mathsf{cfg}}$ holding subcells $\langle ... \rangle_{\mathsf{code}}$, $\langle ... \rangle_{\mathsf{heap}}$, $\langle ... \rangle_{\mathsf{in}}$, $\langle ... \rangle_{\mathsf{out}}$ among many others, holding the current code fragment, heap, input buffer, output buffer, respectively.

- C. Ellison and G. Rosu. An executable formal semantics of C with applications. In POPL, pages 533544. ACM, 2012.
- C. Hathhorn, C. Ellison, and G. Ro³u. Defining the undefinedness of C. In PLDI'15, pages 336345. ACM, 2015.
- A. Stefanescu, D. Park, S. Yuwen, Y. Li, and G. Ro³u. Semantics-based program verifiers for all languages. In OOPSLA'16, pages 7491. ACM, 2016.

Reading, storing and reverse writing a sequence of integers

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

void list_read_write(int n) {
```

rule $\langle \$ \Rightarrow \texttt{return;} \ \cdots \rangle_{\mathsf{code}} \ \langle \mathsf{A} \Rightarrow \cdot \ \cdots \rangle_{\mathsf{in}} \ \langle \cdots \ \cdot \Rightarrow \mathsf{rev}(\mathsf{A}) \rangle_{\mathsf{out}} \ \wedge \ \mathtt{n} = \mathsf{len}(\mathsf{A})$

```
  int i=0;
  struct listNode *x=0;
```

inv $\langle \beta \ \wedge \ \mathsf{len}(\beta) = \mathtt{n} - \mathtt{i} \ \wedge \ \mathtt{i} \leq \mathtt{n} \ \cdots \rangle_{\mathsf{in}} \ \langle \mathsf{list}(\mathtt{x}, \alpha) \ \cdots \rangle_{\mathsf{heap}} \ \wedge \ \mathsf{A} = \mathsf{rev}(\alpha) @ \beta$

```
  while (i < n) {
    struct listNode *y = x;
    x = (struct listNode*) malloc(sizeof(struct listNode));
    scanf("%d", &(x->val));
    x->next = y;
    i += 1; }
```

inv $\langle \cdots \ \alpha \rangle_{\mathsf{out}} \ \langle \mathsf{list}(\mathtt{x}, \beta) \ \cdots \rangle_{\mathsf{heap}} \ \wedge \ \mathsf{rev}(\mathsf{A}) = \alpha @ \beta$

```
  while (x) {
    struct listNode *y;
    y = x->next;
    printf("%d␣",x->val);
    free(x);
    x = y; }
}
```

Towards the loop invariant

$$\langle \langle \texttt{LOOP}\, k \rangle_{\mathsf{code}}\ \langle \texttt{x} \mapsto x,\ \texttt{n} \mapsto n,\ \texttt{i} \mapsto i,\ e \rangle_{\mathsf{env}}\ \langle x \mapsto a,\ x+1 \mapsto y,\ h \rangle_{\mathsf{heap}}\ \langle \beta \rangle_{\mathsf{in}}\ \langle \epsilon \rangle_{\mathsf{out}} \rangle_{\mathsf{cfg}}$$

The intuition for this pattern is that it matches all the configurations whose code starts with `LOOP` (k, the "code frame", matches the rest of the code), whose environment binds program identifiers `n` and `i` to values n and i, respectively, and `x` to location x (e, the "environment frame", matches the rest of the environment map) such that both x and $x+1$ are allocated and bound to some values in the heap (h, the "heap frame", matches the rest of the heap), whose input buffer contains some sequence (β) and whose output buffer contains the empty sequence.

Towards the loop invariant (cont.)

The interesting patterns are those combining symbols and logical connectives. For example, suppose that we want to restrict the pattern above to only match where $i \leq n$.

$$\langle\langle \texttt{LOOP}\, k\rangle_{\mathsf{code}}\ \langle \mathtt{x} \mapsto x,\ \mathtt{n} \mapsto n,\ \mathtt{i} \mapsto i,\ e\rangle_{\mathsf{env}}\ \langle x \mapsto a,\ x+1 \mapsto y,\ h\rangle_{\mathsf{heap}}\ \langle\beta\rangle_{\mathsf{in}}\ \langle\epsilon\rangle_{\mathsf{out}}\rangle_{\mathsf{cfg}}\ \wedge\ i \leq n$$

Quantifiers can be used, for example, to abstract away irrelevant parts of the pattern. Suppose, for example, that we work in a context where the code and the output cells are irrelevant, and so are the frames of the environment and heap cells. Then we can "hide" them to the context as follows:

$$(\exists c\,.\,\exists e\,.\,\exists h\,.\,\langle\langle \mathtt{x} \mapsto x,\ \mathtt{n} \mapsto n,\ \mathtt{i} \mapsto i,\ e\rangle_{\mathsf{env}}\ \langle x \mapsto a,\ x+1 \mapsto y,\ h\rangle_{\mathsf{heap}}\ \langle\beta\rangle_{\mathsf{in}}\ c\rangle_{\mathsf{cfg}})\ \wedge\ i \leq n$$

Following a notational convention proposed and implemented in $\mathbb{K}$ http://kframework.org we use "..." as syntactic sugar for such existential quantifiers used for framing:

$$\langle\langle \mathtt{x} \mapsto x,\ \mathtt{n} \mapsto n,\ \mathtt{i} \mapsto i\ \cdots\rangle_{\mathsf{env}}\ \langle x \mapsto a,\ x+1 \mapsto y\ \cdots\rangle_{\mathsf{heap}}\ \langle\beta\rangle_{\mathsf{in}}\ \cdots\rangle_{\mathsf{cfg}}\ \wedge\ i \leq n$$

Towards the loop invariant (cont.)

Inspired from the invariant of the first loop in slide 5 let us add some more constraints:

$$\langle\langle list(x,\alpha) \ \cdots\rangle_{\mathsf{heap}} \ \langle \beta \ \cdots\rangle_{\mathsf{in}} \ \cdots\rangle_{\mathsf{cfg}} \ \wedge \ \mathsf{len}(\beta) = n - i \ \wedge \ i \leq n \ \wedge \ A = \mathsf{rev}(\alpha)@\beta$$

The pattern above is additionally stating that the $\langle ... \rangle_{\mathsf{in}}$ cell starts with a prefix of size equal to $n - i$ which appended to the reverse of the sequence that x points to in the heap equals the original input sequence A. We can arrange the pattern to better localize the logical constraints to the sub-patterns for which they are relevant. For example, the first two constraints are relevant for the sequence β, so we can move them to their place:

$$\langle\langle list(x,\alpha) \ \cdots\rangle_{\mathsf{heap}} \ \langle \beta \ \wedge \ \mathsf{len}(\beta) = n - i \ \wedge \ i \leq n \ \cdots\rangle_{\mathsf{in}} \ \cdots\rangle_{\mathsf{cfg}} \ \wedge \ A = \mathsf{rev}(\alpha)@\beta$$

Towards the loop invariant (cont.)

The above transformation is indeed correct, thanks to Proposition about constraint propagation . Similarly, the remaining constraint can be localized to the two cells that need it. Using also the fact that cell concatenation is commutative, we rewrite the pattern into:

$$\langle\langle \beta \ \wedge \ \mathsf{len}(\beta) = n - i \ \wedge \ i \leq n \ \cdots\rangle_{\mathsf{in}} \ \langle \mathit{list}(x, \alpha) \ \cdots\rangle_{\mathsf{heap}} \ \wedge \ A = \mathsf{rev}(\alpha)@\beta \ \cdots\rangle_{\mathsf{cfg}}$$

Syntax of patterns

Definition 2.1. *Let (S, Σ) be a many-sorted signature of* ***symbols****. Matching logic (S, Σ)-* ***formulae****, also called (S, Σ)-* ***patterns****, or just (matching logic)* ***formulae*** *or* ***patterns*** *when (S, Σ) is understood from context, are inductively defined as follows for all sorts $s \in S$:*

$$
\begin{array}{rcll}
\varphi_s & ::= & x \in Var_s & // \ \textit{Variable} \\
& | & \sigma(\varphi_{s_1}, ..., \varphi_{s_n}) \ \textit{with} \ \sigma \in \Sigma_{s_1 ... s_n, s} \quad (\textit{written} \ \Sigma_{\lambda, s} \ \textit{when} \ n = 0) & // \ \textit{Structure} \\
& | & \neg \varphi_s & // \ \textit{Complement} \\
& | & \varphi_s \wedge \varphi_s & // \ \textit{Intersection} \\
& | & \exists x \,.\, \varphi_s \ \textit{with} \ x \in Var \quad (\textit{of any sort}) & // \ \textit{Binding}
\end{array}
$$

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

Some “standard” syntax stuff

To ease notation, $\varphi \in \text{PATTERN}$ means φ is a pattern, while $\varphi_s \in \text{PATTERN}$ or $\varphi \in \text{PATTERN}_s$ that it has sort s. We adopt the following derived constructs (“syntactic sugar”):

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

We adapt from first-order logic the notions of free variable, (variable capture free) substitution, and variable renaming, briefly recalled below. Let $FV(\varphi)$ denote the *free variables* of φ, defined as follows: $FV(x) = \{x\}$, $FV(\sigma(\varphi_{s_1}, ..., \varphi_{s_n})) = FV(\varphi_{s_1}) \cup \cdots \cup FV(\varphi_{s_n})$, $FV(\neg\varphi) = FV(\varphi)$, $FV(\varphi_1 \wedge \varphi_2) = FV(\varphi_1) \cup FV(\varphi_2)$, and $FV(\exists x.\varphi) = FV(\varphi) \setminus \{x\}$.

Semantics

Definition 2.2. *A* ***matching logic*** (S, Σ)*-****model*** M*, or just a* Σ*-****model*** *when* S *is understood, or simply a* ***model*** *when both* S *and* Σ *are understood, consists of:*

(1) *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*

(2) *A function* $\sigma_M : M_{s_1} \times \cdots \times M_{s_n} \to \mathcal{P}(M_s)$ *for each symbol* $\sigma \in \Sigma_{s_1 \ldots s_n, s}$*, called the* ***interpretation*** *of* σ *in* M*.*

Note that symbols are interpreted as relations, and that the usual (S, Σ)-algebra models are a special case 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)-algebra models also fall as special case, where $|\sigma_M(m_1, \ldots, m_n)| \leq 1$, since we can capture the undefinedness of σ_M on $m_1, \ldots, m_n$ with $\sigma_M(m_1, \ldots, m_n) = \emptyset$. We tacitly use the same notation σ_M for its extension to argument sets, $\mathcal{P}(M_{s_1}) \times \cdots \times \mathcal{P}(M_{s_n}) \to \mathcal{P}(M_s)$, that is,

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

where $A_1 \subseteq M_{s_1}, \ldots, A_n \subseteq M_{s_n}$.

Why *matching* logic?

Definition 2.3. *Given a model M and a map $\rho : Var \to M$, called an M-**valuation**, let its extension $\overline{\rho} : \text{PATTERN} \to \mathcal{P}(M)$ be inductively defined as follows:*

- $\overline{\rho}(x) = \{\rho(x)\}$, *for all* $x \in Var_s$
- $\overline{\rho}(\sigma(\varphi_1, \ldots, \varphi_n)) = \sigma_M(\overline{\rho}(\varphi_1), \ldots \overline{\rho}(\varphi_n))$ *for all* $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ *and appropriate* $\varphi_1, \ldots, \varphi_n$
- $\overline{\rho}(\neg\varphi) = M_s \setminus \overline{\rho}(\varphi)$ *for all* $\varphi \in \text{PATTERN}_s$
- $\overline{\rho}(\varphi_1 \wedge \varphi_2) = \overline{\rho}(\varphi_1) \cap \overline{\rho}(\varphi_2)$ *for all* φ_1, φ_2 *patterns of the same sort*
- $\overline{\rho}(\exists x.\varphi) = \bigcup\{\overline{\rho'}(\varphi) \mid \rho' : Var \to M,\ \rho'|_{Var\setminus\{x\}} = \rho|_{Var\setminus\{x\}}\} = \bigcup_{a \in M} \overline{\rho[a/x]}(\varphi)$

*where "$\setminus$" is set difference, "$\rho|_V$" is ρ restricted to $V \subseteq Var$, and "$\rho[a/x]$" is map ρ' with $\rho'(x) = a$ and $\rho'(y) = \rho(y)$ if $y \neq x$. If $a \in \overline{\rho}(\varphi)$ then we say a **matches** φ (with witness ρ).*

How to understand *matching*?

It is easy to see that the usual notion of term matching is an instance of the above; indeed, if φ is a term with variables and M is the ground term model, then a ground term a matches φ iff there is some substitution ρ such that $\rho(\varphi) = a$. It may be insightful to note that patterns can also be regarded as predicates, when we think of "a matches pattern φ" as "predicate φ holds in a". But matching logic allows more complex patterns than terms or predicates, and models which are not necessarily conventional (term) algebras.

Interpreting formulae as sets of elements in models is reminiscent of modal logic, where they are interpreted as the "worlds" in which they hold, and of separation logic, where they are interpreted as the "heaps" they match.

Extension works as expected with the derived constructs

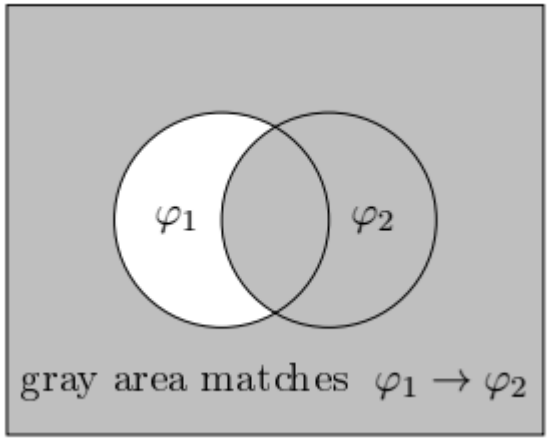
- $\overline{\rho}(\top_s) = M_s$ and $\overline{\rho}(\bot_s) = \emptyset$
- $\overline{\rho}(\varphi_1 \vee \varphi_2) = \overline{\rho}(\varphi_1) \cup \overline{\rho}(\varphi_2)$
- $\overline{\rho}(\varphi_1 \rightarrow \varphi_2) = \{m \in M_s \mid m \in \overline{\rho}(\varphi_1) \text{ implies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

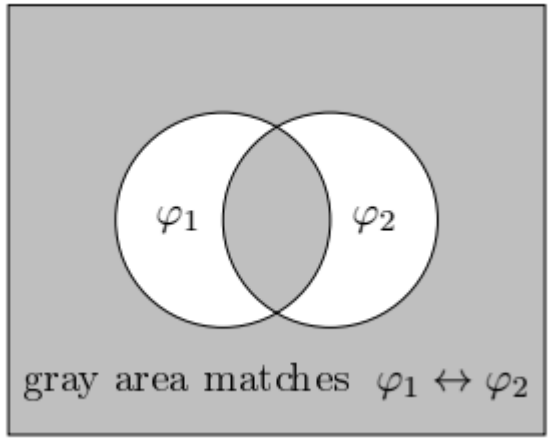
Warning: Matching Logic is not two-valued!

Therefore, the matching logic interpretation of the logical connectives is not two-valued like in classical logics. In particular, the interpretation of $\varphi_1 \to \varphi_2$ is the set of all the elements that if matched by φ_1 then are also matched by φ_2. One should be careful when reasoning with such non-classical logics, as basic intuitions may deceive. For example, the interpretation of $\varphi_1 \to \varphi_2$ is the total set (i.e., same as $\top$) iff all elements matching φ_1 also match φ_2, but it is the empty set iff φ_2 is matched by no elements (same as $\bot$) while φ_1 is matched by all elements (same as $\top$).

Matching Logic is not two-valued! (cont.)

If in doubt, thanks to the set-theoretical interpretation of the matching logic connectives, we can always draw diagrams to enhance our intuition; for example, Figure depicts the semantics of pattern implication and of pattern equivalence.





- $\overline{\rho}(\varphi_1 \to \varphi_2) = \{m \in M_s \mid m \in \overline{\rho}(\varphi_1) \text{ implies } m \in \overline{\rho}(\varphi_2)\} = M_s \setminus (\overline{\rho}(\varphi_1) \setminus \overline{\rho}(\varphi_2))$
- $\overline{\rho}(\varphi_1 \leftrightarrow \varphi_2) = \{m \in M_s \mid m \in \overline{\rho}(\varphi_1) \text{ iff } m \in \overline{\rho}(\varphi_2)\} = M_s \setminus (\overline{\rho}(\varphi_1) \ \Delta \ \overline{\rho}(\varphi_2))$

Predicates

Definition 2.4. *Pattern φ_s is an M-**predicate**, or a **predicate in** M, iff for any M-**valuation** $\rho : Var \to M$, it is the case that $\overline{\rho}(\varphi_s)$ is either M_s (it holds) or $\emptyset$ (it does not hold). Pattern φ_s is a **predicate** iff it is a predicate in all models M.*

Note that $\top_s$ and $\bot_s$ are predicates, and if φ, φ_1 and φ_2 are predicates then so are $\neg\varphi$, $\varphi_1 \wedge \varphi_2$, and $\exists x \,.\, \varphi$. That is, the logical connectives of matching logic preserve the predicate nature of patterns.

Definition 2.5. *M **satisfies** φ_s, written $M \models \varphi_s$, iff $\overline{\rho}(\varphi_s) = M_s$ for all $\rho : Var \to M$.*

Properties of entailment

Proposition 2.6. *Unless otherwise stated, assume the default pattern sort to be s. Then:*

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

(2) *If* $x \in Var_s$ *then* $M \models x$ *iff* $|M_s| = 1$

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

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

(5) $M \models \varphi_1 \wedge \varphi_2$ *iff* $M \models \varphi_1$ *and* $M \models \varphi_2$

(6) *If* $\exists x.\varphi_s$ *closed,* $M \models \exists x.\varphi_s$ *iff* $\bigcup\{\overline{\rho}(\varphi_s) \mid \rho : Var \to M\} = M_s$; *hence,* $M \models \exists x.x$

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

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

(9) $M \models \forall x.\varphi$ *iff* $M \models \varphi$

Warning on entailment

Note that property "if φ closed then $M \models \neg\varphi$ iff $M \not\models \varphi$", which holds in classical logics like FOL, does not hold in matching logic. This is because $M \models \neg\varphi$ means $\neg\varphi$ is matched by all elements, i.e., φ is matched by no element, while $M \not\models \varphi$ means φ is not matched by some elements. These two notions are different when patterns can have more than two interpretations, which happens when M can have more than one element.

Matching logic specification

Definition 2.7. *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 for each M, $M \models F$ implies $M \models \varphi$. A* ***matching logic specification*** *is a triple (S, Σ, F) with $F \subseteq$ PATTERN.*

Pure predicate logic reasoning can be used to reason about patterns

Proposition 2.8. *The following properties hold for patterns of any sort $s \in S$, so the Hilbert-style axioms and proof rules that are sound and complete for pure predicate logic are also sound for matching logic, for any sort*

(1) $\models \varphi$, *where* φ *is a propositional tautology over patterns of sort* s.

(2) *Modus ponens:* $\models \varphi_1$ *and* $\models \varphi_1 \to \varphi_2$ *imply* $\models \varphi_2$.

(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* $\models \forall x \,.\, \varphi$.

(5) *Substitution:* $\models (\forall x \,.\, \varphi) \to \varphi[y/x]$, *with variable* $y \notin FV(\forall x \,.\, \varphi)$ *of same sort as* x.

Basic structural properties

Proposition 2.10. ***(Structural Framing)*** *If* $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ *and* $\varphi_i, \varphi_i' \in \text{PATTERN}_{s_i}$ *such that* $\models \varphi_i \to \varphi_i'$ *for all* $i \in 1 \ldots n$, *then* $\models \sigma(\varphi_1, \ldots, \varphi_n) \to \sigma(\varphi_1', \ldots, \varphi_n')$.

Proposition 2.11. ***(Distributivity of symbol application)*** *Let* $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ *and* $\varphi_i \in \text{PATTERN}_{s_i}$ *for all* $1 \leq i \leq n$. *Pick a particular* $1 \leq i \leq n$. *Let* $\varphi_i' \in \text{PATTERN}_{s_i}$ *be another pattern of sort* s_i *and let* $C_{\sigma,i}[\Box]$ *be the context* $\sigma(\varphi_1, \ldots, \varphi_{i-1}, \Box, \varphi_{i+1}, \ldots \varphi_n)$ *(a context* $C[\Box]$ *is a pattern with one occurrence of a free variable, "*$\Box$*", and* $C[\varphi]$ *is* $C[\varphi/\Box]$*). Then:*

(1) $\models C_{\sigma,i}[\varphi_i \vee \varphi_i'] \leftrightarrow C_{\sigma,i}[\varphi_i] \vee C_{\sigma,i}[\varphi_i']$
(2) $\models C_{\sigma,i}[\exists x \,.\, \varphi_i] \leftrightarrow \exists x \,.\, C_{\sigma,i}[\varphi_i]$, *where* $x \notin FV(C_{\sigma,i}[\Box])$
(3) $\models C_{\sigma,i}[\varphi_i \wedge \varphi_i'] \to C_{\sigma,i}[\varphi_i] \wedge C_{\sigma,i}[\varphi_i']$
(4) $\models C_{\sigma,i}[\forall x \,.\, \varphi_i] \to \forall x \,.\, C_{\sigma,i}[\varphi_i]$, *where* $x \notin FV(C_{\sigma,i}[\Box])$

Basic structural properties (cont.)

Definition 2.12. *With the notation in Proposition 2.11, $C_{\sigma,i}[\Box]$ is* ***injective*** *in specification (S, Σ, F) iff $F \models C_{\sigma,i}[x] \wedge C_{\sigma,i}[y] \rightarrow C_{\sigma,i}[x \wedge y]$, where $x, y \in Var_{s_i}$ are distinct variables which do not occur in $C_{\sigma,i}[\Box]$. We drop (S, Σ, F) when understood. Symbol σ is* ***injective on position*** *i iff $C_{\sigma,i}[\Box]$ is injective with φ_1, ..., φ_{i-1}, φ_{i+1},...,φ_n chosen as distinct variables.*

Proposition 2.13. ***(Distributivity of injective symbol application)*** *With the notation in Definition 2.12, if $C_{\sigma,i}[\Box]$ is injective in (S, Σ, F) and $\varphi_i, \varphi_i' \in \text{PATTERN}_{s_i}$ then:*

(1) $F \models C_{\sigma,i}[\varphi_i] \wedge C_{\sigma,i}[\varphi_i'] \rightarrow C_{\sigma,i}[\varphi_i \wedge \varphi_i']$

(2) $F \models \forall x \,.\, C_{\sigma,i}[\varphi_i] \rightarrow C_{\sigma,i}[\forall x \,.\, \varphi_i]$, *where* $x \notin FV(C_{\sigma,i}[\Box])$

Together with Proposition 2.11, this implies the full distributivity of injective contexts w.r.t. the matching logic constructs $\wedge$, $\vee$, $\forall$, $\exists$ (but not $\neg$).

Instances

October 28, 2022

Propositional Calculus

On slide 22 (1) in Proposition 2.8, we showed that propositional reasoning is sound for matching logic. Here we go one step further and show that we can can instantiate matching logic to become precisely propositional calculus, without any translation needed in any direction. The idea is to add a special sort for propositions, say *Prop*, then to use the already existing syntax of matching logic to build propositions as we know them, and then to show that the existing semantics of matching logic, given by $\models$, yields the expected semantics of propositions as we know it in propositional calculus (let us refer to it as $\models_{Prop}$).

We build a matching logic signature as follows: S contains only one sort, *Prop*, and Σ is empty. Let us also drop the existential quantifier, so that the resulting syntax of patterns becomes exactly that of propositional calculus: $\varphi ::= Var_{Prop} \mid \neg\varphi \mid \varphi \wedge \varphi$.

Proposition 3.1. *For any proposition φ, the following holds:* $\models_{Prop} \varphi$ *iff* $\models \varphi$.

(Pure) Predicate Logic

Recall , that by *pure* predicate logic in this paper we mean predicate logic or first-order logic (FOL) with only predicate symbols (no function and no constant symbols).

Proposition 2.8 showed that predicate logic reasoning is sound for matching logic. Similarly to propositional calculus in Slide 26 , here we go one step further and show that we can can instantiate matching logic to become precisely predicate logic

We follow the same approach like for propositional calculus: add a special sort for predicates, say *Pred*, then use the already existing syntax of matching logic to build formulae as we know them in predicate logic, and then show that the existing semantics of matching logic, given by $\models$, yields the expected semantics of pure predicate logic. We let $\models_{PL}$ denote the predicate logic satisfaction.

(Pure) Predicate Logic (cont.)

Recall that predicate logic is the fragment of first-order logic with just predicate symbols, that is, with no function (including no constant) and no equality symbols. We consider only the many-sorted case here. Formally, if S is a sort set and Π is a set of predicate symbols, the syntax of pure predicate logic formulae is

$$\varphi \quad ::= \quad \pi(x_1, \ldots, x_n) \text{ with } \pi \in \Pi_{s_1 \ldots s_n},\ x_1 \in Var_{s_1},\ \ldots,\ x_n \in Var_{s_n}$$
$$\mid \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\, \varphi\ .$$

Without loss of generality, suppose that we can pick a fresh sort name, *Pred*; that is, $Pred \notin S$. Let us now construct the matching logic signature $(S \cup \{Pred\}, \Sigma)$, where $\Sigma_{s_1 \ldots s_n, Pred} = \Pi_{s_1 \ldots s_n}$ are the only symbols in Σ; that is, Σ contains precisely the predicate symbols of the predicate logic signature, but regarded as pattern symbols of result sort *Pred*. Suppose also that we disallow any variables of sort *Pred* in patterns. Then the matching logic patterns of sort *Pred* are precisely the predicate logic formulae, without any translation in any direction.

(Pure) Predicate Logic (cont.)

Proposition 4.1. *For any predicate logic formula φ, the following holds: $\models_{PL} \varphi$ iff $\models \varphi$.*

Proof. That $\models_{PL} \varphi$ implies $\models \varphi$ follows by Proposition 2.8: each of the proof rules of the complete proof system of (pure) predicate logic is sound for matching logic. For the other implication, note that we can associate to any predicate logic model $M^{PL} = (\{M_s^{PL}\}_{s \in S}, \{\pi_{M^{PL}}\}_{\pi \in \Pi})$ a matching logic model $M^{ML} = (\{M_s^{ML}\}_{s \in S \cup \{Pred\}}, \{\pi_{M^{ML}}\}_{\pi \in \Sigma})$, where $M_s^{ML} = M_s^{PL}$ for all $s \in S$ and $M_{Pred}^{ML} = \{\star\}$ (with $\star$ some arbitrary but fixed element) and $\pi_{M^{ML}}(a_1, \ldots, a_n) = \{\star\}$ iff $\pi_{M^{PL}}(a_1, \ldots, a_n)$ holds, and $\pi_{M^{ML}}(a_1, \ldots, a_n) = \emptyset$ otherwise. Furthermore, we can show that for any PL formula φ, we have $M^{PL} \models_{PL} \varphi$ iff $M^{ML} \models_{ML} \varphi$. Since φ does not contain any variables of sort *Pred*, by (1) in Proposition 2.6 it suffices to show that for any $\rho : Var \to M^{PL}$, it is the case that $M^{PL}, \rho \models_{PL} \varphi$ iff $\overline{\rho}(\varphi) = \{\star\}$. We can easily show this property by structural induction on φ. The only relatively non-trivial case is the complement construct, which shows why it was important for M_{Pred}^{ML} to contain precisely one element: $M^{PL}, \rho \models_{PL} \neg\varphi$ iff $M^{PL}, \rho \not\models_{PL} \varphi$ iff (by the induction hypothesis) $\overline{\rho}(\varphi) \neq \{\star\}$ iff $\overline{\rho}(\varphi) = \emptyset$ iff $\overline{\rho}(\neg\varphi) = \{\star\}$.

Therefore, $M^{PL} \models_{PL} \varphi$ iff $M^{ML} \models_{ML} \varphi$. Since the predicate logic model M^{PL} was chosen arbitrarily, it follows that $\models \varphi$ implies $\models_{PL} \varphi$. □

First-Order Logic

Formally, given a FOL signature (S, Σ, Π), the syntax of its (many-sorted) formulae is:

$$
\begin{array}{rcl}
t_s & ::= & x \in Var_s \\
 & | & \sigma(t_{s_1}, \ldots, t_{s_n}) \text{ with } \sigma \in \Sigma_{s_1 \ldots s_n, s} \\
\varphi & ::= & \pi(t_{s_1}, \ldots, t_{s_n}) \text{ with } \pi \in \Pi_{s_1 \ldots s_n} \\
 & | & \neg\varphi \mid \varphi \wedge \varphi \mid \exists x.\, \varphi \,.
\end{array}
$$

Compare the above with the syntax of matching logic in Slide 10. Unlike FOL, matching logic does not distinguish between the term and predicate syntactic categories, reason for which its syntax is in fact more compact than FOL's. Moreover, matching logic allows logical constructs over all the syntactic categories, not only over predicates, and locally where they are needed instead of only at the top, predicate level. Also, matching logic allows quantification over any sorts, including over sorts of symbols thought of as predicates.

First-Order Logic (cont.)

Like in predicate logic, we add a *Pred* sort and regard the FOL predicate symbols as matching logic symbols of result *Pred*, and disallow variables of sort *Pred* and restrict the use of logical connectives and quantifiers to only patterns of sort *Pred*. Then there is a one-to-one correspondence between FOL formulae and matching logic patterns of sort *Pred*; we let φ range over them. Moreover,
we constrain each FOL operational symbol $\sigma \in \Sigma_{s_1 \ldots s_n, s}$ to be interpreted as a function, that is, we write the symbols meant to be functions as $\sigma : s_1 \ldots s_n \to s$. Formally, let (S^{ML}, Σ^{ML}) be the matching logic signature with $S^{ML} = S \cup \{Pred\}$ and $\Sigma^{ML} = \Sigma \cup \{\pi : s_1 \ldots s_n \to Pred \mid \pi \in \Pi_{s_1 \ldots s_n}\}$, and let F be $\{\exists z{:}s \,.\, \sigma(x_1{:}s_1, \ldots, x_n{:}s_n) = z \mid \sigma \in \Sigma_{s_1 \ldots s_n, s}\}$ stating that each symbol in Σ is a function.

Proposition 7.1. *For any FOL formula φ, we have $\models_{FOL} \varphi$ iff $F \models \varphi$.*

First-Order Logic (cont.)

Consider we want to refer to all real numbers of the form $1/x$ with x a strictly positive integer, but this time using a given predicate $positive?(x)$ that tells whether x is positive. We can use the pattern $1/x \wedge (positive?(x) =^{Real}_{Pred} \top_{Pred})$, but that is too verbose. We would like to just write $1/x \wedge positive?(x)$.

Notation 7.2. *If φ is a FOL formula, we take the freedom to write φ instead of $\varphi = \top_{Pred}$.*

The following result allows us to do that:

Proposition 7.3. *If p, p_1 and p_2 are FOL formulae, then*

- $\models (p_1 = \top_{Pred} \wedge p_2 = \top_{Pred}) = (p_1 \wedge p_2 = \top_{Pred})$
- $\models \neg(p = \top_{Pred}) = (\neg p = \top_{Pred})$

Other similar properties for derived FOL constructs can be derived from these.

Proof. Trivial: each of $\overline{\rho}(p)$, $\overline{\rho}(p_1)$, and $\overline{\rho}(p_2)$ can only be $\emptyset$ or $\{\star\}$, for any valuation ρ. □

Modal Logic

Hereafter we only discuss S5 and thus implicitly mean the S5 modal logic whenever we say modal logic.

We start by giving the syntax and semantics of modal logic. Let Var_{Prop} be a countable set of *propositional variables* p, q, etc. Then the modal logic syntax is defined as follows:

$$\begin{aligned} \varphi \quad ::= \quad & Var_{Prop} \\ & \mid \neg\varphi \mid \varphi \rightarrow \varphi \mid \Box\varphi \,. \end{aligned}$$

The remaining propositional constructs $\wedge$, $\vee$ and $\leftrightarrow$, can be defined as derived constructs. Therefore, syntactically, modal logic adds the $\Box$ construct to propositional logic, which is called *necessity*: $\Box\varphi$ is read "it is necessary that φ". The dual *possibility* construct can be defined as a derived construct: $\Diamond\varphi \equiv \neg\Box\neg\varphi$ is read "it is possible that φ". Semantically, the truth value of a formula is relative to a "world". Propositions can be true in some worlds but false in others, and thus formulae can also be true in some worlds but not in others:

Modal Logic (cont.)

Definition 8.1. *Let W be a set of* ***worlds****. Mappings $v : Var_{Prop} \times W \to \{true, false\}$, called* ***(modal logic) W-valuations****, state that each proposition only holds in a given (possibly empty or total) subset of worlds. Valuations extend to modal logic formulae:*

- $v(\neg\varphi, w) = true$ *iff* $v(\varphi, w) = false$
- $v(\varphi_1 \to \varphi_2, w) = true$ *iff* $v(\varphi_1, w) = false$ *or* $v(\varphi_2, w) = true$
- $v(\Box\varphi, w) = true$ *iff* $v(\varphi, w') = true$ *for every* $w' \in W$

Formula φ is ***valid in*** *W, written $W \models_{S5} \varphi$, iff $v(\varphi, w) = true$ for any W-valuation v and any $w \in W$. Formula φ is* ***valid****, written $\models_{S5} \varphi$, iff $W \models_{S5} \varphi$ for all W.*

Modal logic (S5) admits the following sound and complete proof system

(N) Rule: If φ derivable then $\Box\varphi$ derivable

(What about Modus Ponens Rule: If φ and $\varphi \to \psi$ are derivable then ψ is derivable?

(K) Axiom: $\Box(\varphi_1 \to \varphi_2) \to (\Box\varphi_1 \to \Box\varphi_2)$

(M) Axiom: $\Box\varphi \to \varphi$

(5) Axiom: $\Diamond\varphi \to \Box\Diamond\varphi$

Modal Logic (cont.)

We next show that we can define a matching logic specification (S, Σ, F) which faithfully captures modal logic, both syntactically and semantically. The idea is quite simple: S contains precisely one sort, say $World$; Σ contains one constant symbol $p \in \Sigma_{\lambda, World}$ for each propositional variable $p \in Var_{Prop}$, plus a unary symbol $\Diamond \in \Sigma_{World, World}$; and F contains precisely one axiom stating that $\Diamond$ is the definedness symbol , namely $\Diamond x : World$ (x is a free $World$ variable in this pattern). Then we let $\Box\varphi$ be the totality construct , that is, syntactic sugar for $\neg\Diamond\neg\varphi$.

Modal Logic (cont.)

Note that any modal logic formula φ can be regarded, as is, as a ground matching logic pattern over this signature; by "ground" we mean a pattern without variables, so the other implication is also true, because disallowing variables includes disallowing quantifiers.

Proposition 8.2. *For any modal logic formula φ, we have $\models_{\mathrm{S5}} \varphi$ iff $\models \varphi$.*

Separation Logic

Separation logic (originating with ideas in late 1990, followed by a *canonical* work

- J. C. Reynolds. Separation logic: A logic for shared mutable data structures. In LICS'02, pages 55-74. IEEE, 2002.

There are many variants, but here we only consider the original variant according to the cited paper. 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.

Separation logic extends the syntax of formulae in FOL As follows:

$$
\begin{array}{rcll}
\varphi & ::= & (FOL\ syntax) & \\
& | & emp & \\
& | & Nat \mapsto Nat & \\
& | & \varphi * \varphi & \\
& | & \varphi -\!\!* \varphi & \text{“magic wand”}
\end{array}
$$

Separation Logic (cont.)

Its semantics is based on a fixed model of stores and heaps, which are finite-domain maps from variables and, respectively, locations (which are particular numbers), to integers. Below we recall the formal definition of satisfaction in the original variant of separation logic, noting that subsequent variants of separation logic extend the underlying model to include stacks (instead of stores) as well as various types of resources that are encountered in modern programming languages.

Nevertheless, we leave the thorough analysis of the diversity of separation logic variants proposed in the last 15 years through the lenses of matching logic as a subject for future work.

Separation Logic (cont.)

Definition 9.1. *(Separation logic semantics) Partial finite-domain maps $s : Var \rightharpoonup Nat$ are called* ***stores****, partial finite-domain maps $h : Nat \rightharpoonup Nat$ are called* ***heaps****, and pairs (s, h) of a store and a heap are called* ***states****. The semantics of the separation logic constructs are given in terms of such states, as follows:*

- $(s, h) \models_{SL} \varphi$ *for a FOL formula* φ *iff* $s \models_{FOL} \varphi$ *(the heap portion of the model is irrelevant for the FOL fragment);*
- $(s, h) \models_{SL} emp$ *iff* $Dom(h) = \emptyset$*;*

Separation Logic (cont.)

Definition 9.1. *(Separation logic semantics, cont.)*

- $(s,h) \models_{SL} e_1 \mapsto e_2$ *where* e_1 *and* e_2 *are terms of sort Nat (thought of as "expressions") iff* $Dom(h) = \overline{s}(e_1) \neq 0$ *and* $h(\overline{s}(e_1)) = \overline{s}(e_2)$*, where* $\overline{s}$ *is the (partial function) extension of* s *to expressions (with variables) of sort Nat, defined similarly to the extension of valuations to patterns in Definition 2.3* (slide 13);
- $(s,h) \models_{SL} \varphi_1 * \varphi_2$ *iff there exist* h_1 *and* h_2 *such that* $Dom(h_1) \cap Dom(h_2) = \emptyset$ *and* $h = h_1 * h_2$ *(the merge of* h_1 *and* h_2*, a partial function on maps written as an associative/commutative* monoid) *and* $(s,h_1) \models_{SL} \varphi_1$, $(s,h_2) \models_{SL} \varphi_2$;
- $(s,h) \models_{SL} \varphi_1 -\!\!* \varphi_2$ *iff for any* h_1 *with* $Dom(h_1) \cap Dom(h) = \emptyset$*, if* $(s,h_1) \models_{SL} \varphi_1$ *then* $(s, h * h_1) \models_{SL} \varphi_2$*; i.e., the semantics of "magic wand" is defined as the states whose heaps extended with a fragment satisfying* φ_1 *result in ones satisfying* φ_2.

Separation logic formula φ *is* ***valid****, written* $\models_{SL} \varphi$*, iff* $(s,h) \models_{SL} \varphi$ *for any state* (s,h).

Separation Logic (cont.)

One of the most appealing aspects of separation logic is that it allows us to define compact and elegant specifications of heap abstractions using inductively defined predicates. Such an abstraction which is quite common is the linked-list abstraction $list(x, S)$ stating that x points to a linked list containing an abstract sequence of natural numbers S:

$$\begin{aligned} list(x, \epsilon) &\stackrel{def}{=} emp \wedge x = 0 \\ list(x, n \cdot S) &\stackrel{def}{=} \exists z \,.\, x \mapsto [n, z] * list(z, S) \end{aligned}$$

Above, ϵ is the empty sequence, $n \cdot S$ is the sequence starting with natural number n and followed by sequence S, and $x \mapsto [n, z]$ is syntactic sugar for $x \mapsto n * (x + 1) \mapsto z$. So a linked list starting with address x takes either empty heap space, in which case x must be 0 and the abstracted sequence is ϵ, or there is some node in the linked list at location x in the heap that holds the head of the abstracted sequence (n) and a link (z) to another linked list that holds the tail of the abstracted sequence (S).

Separation Logic (cont.)

We only discuss maps from natural numbers to natural numbers, but they can be similarly defined over arbitrary domains as keys and as values. Consider a matching logic specification of maps with its axioms explicitly listed, and with a syntax that deliberately resembles that of separation logic (i.e., we use "*" instead of ",")

$$\_ \mapsto \_ : Nat \times Nat \rightharpoonup Map \qquad emp * H = H$$
$$emp : \rightarrow Map \qquad H_1 * H_2 = H_2 * H_1$$
$$\_ * \_ : Map \times Map \rightharpoonup Map \qquad (H_1 * H_2) * H_3 = H_1 * (H_2 * H_3)$$
$$0 \mapsto a = \bot \qquad x \mapsto a * x \mapsto b = \bot$$

Recall that there are no predicates here, only patterns. When regarding the above ADT as a matching logic specification, we can prove that the bottom two pattern equations above are equivalent to $\neg(0 \mapsto a)$ and, respectively, $(x \mapsto a * y \mapsto b) \rightarrow x \neq y$, giving the $\_ \mapsto \_$ and $\_ * \_$ the feel of "predicates".

Separation Logic (cont.)

Maps, like natural numbers, do not admit finite (or even recursively enumerable) equational (or first-order) axiomatizations, so adding a good enough subset of equations is the best we can do in practice. We chose ones that have been proposed by algebraic specification languages and by separation logics for several reasons.

- First, they have been extensively used, so there is a good chance they are good enough for many purposes.
- Second, we do not want to imply that we propose a novel axiomatization of maps; our novelty is the presentation of known specifications of maps using the general infrastructure of matching logic at no additional translation cost.

Separation Logic (cont.)

Consider the canonical model of partial maps 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 $\rightharpoonup$). M satisfies all axioms above.

Separation Logic (cont.)

We start with matching logic definitions for complete linked lists and for list fragments. Let $list \in \Sigma_{Nat,Map}$ and $lseg \in \Sigma_{Nat \times Nat,Map}$ be two symbols together with patterns

$$list(0) = emp \qquad lseg(x, x) = emp$$
$$list(x) \wedge x \neq 0 = \exists z \,.\, x \mapsto z * list(z) \qquad lseg(x, y) \wedge x \neq y = \exists z \,.\, x \mapsto z * lseg(z, y)$$

The main difference between our definitions above and their separation logic variants is that the latter cannot use the (FOL) equality symbol as we did.

There are two important questions about the matching logic specification above:

(1) Does this specification admit any solution in M, i.e., total relations $list_M : M_{Nat} \to \mathcal{P}(M_{Map})$ and $lseg_M : M_{Nat} \times M_{Nat} \to \mathcal{P}(M_{Map})$ satisfying the patterns above?

(2) If yes, is the solution unique? This is particularly important because we do not require initiality constraints on M nor smallest fixed-point constraints on solutions.

We answer these questions positively.

Separation Logic as an Instance of Matching Logic

Consider the FOL fragment in Slides 30-32 where the signature Σ includes the signature of maps . Any additional FOL constructs, background theories, and/or built-in domains that one wants to consider in separation logic specifications, are handled as already explained in Slides 30-32

It is clear then that all the syntactic constructs of separation logic, except for the magic wand, $-\!\!*$, are given by the above matching logic signature. The magic wand, on the other hand, can be defined as the following derived construct:

$$\varphi_1 -\!\!* \varphi_2 \quad \equiv \quad \exists H : Map \,.\, H \wedge \lfloor H * \varphi_1 \rightarrow \varphi_2 \rfloor$$

Recall that $\lfloor \varphi \rfloor$ is $\top$ (it matches the entire set) iff its enclosed pattern φ is $\top$; otherwise, if φ does not match some elements, then $\lfloor \varphi \rfloor$ is $\bot$ (it matches nothing). In words, $\varphi_1 -\!\!* \varphi_2$ matches all maps h which merged with maps matching φ_1 yield only maps matching φ_2.

Separation Logic as an Instance of Matching Logic

Thanks to the notational convention that Booleans b, respectively usual predicates p, stand for equalities $b = \mathit{true}$, respectively $p = \top_{Pred}$,

Any separation logic formula is a matching logic pattern of sort Map.

Semantically, note that separation logic hard-wires a particular model of maps. That is, its satisfaction relation $\models_{SL} \varphi$ is defined using a pre-defined universe of maps, which is conceptually the same as our model of maps

Proposition 9.2. *If φ is a separation logic formula, then $\models_{SL} \varphi$ iff $Map \models \varphi$.*

Reduction to FOL with equality and axiomatization

November 1, 2022

Translation of FOL to Pure Predicate Logic

It is known that FOL formulae can be translated into equivalent predicate logic with equality formulae (i.e., no function or constant symbols) , by replacing all functions with their graph relations . Specifically, function symbols $\sigma : s_1 \times \cdots \times s_n \to s$ are replaced with predicate symbols $\pi_\sigma : s_1 \times \cdots \times s_n \times s$, and then terms are transformed into formulae by adding existential quantifiers for subterms. Let us define such a translation, say PL. It takes each FOL predicate $\pi(t_1, \ldots, t_n)$ into a pure predicate logic formula as follows:

$$PL(\pi(t_1, \ldots, t_n)) = \exists r_1 \cdots r_n \,.\, PL_2(t_1, r_1) \wedge \cdots \wedge PL_2(t_n, r_n) \wedge \pi(r_1, \ldots, r_n)$$

where $PL_2(t, r)$ is a translation of term t into a predicate stating that t equals variable r:

$$\begin{aligned} PL_2(x, r) &= (x = r) \\ PL_2(\sigma(t_1, \ldots, t_n), r) &= \exists r_1 \cdots \exists r_n \,.\, PL_2(t_1, r_1) \wedge \cdots \wedge PL_2(t_n, r_n) \wedge \pi_\sigma(r_1, \ldots, r_n, r) \end{aligned}$$

Translation of Matching Logic

We can similarly translate matching logic patterns into equivalent predicate logic formulae. Consider predicate logic with equality (and no function or constant symbols) whose satisfaction relation is $\models^{=}_{PL}$. For matching logic signature (S, Σ), let (S, Π_Σ) be the predicate logic signature with $\Pi_\Sigma = \{\pi_\sigma : s_1 \times \cdots \times s_n \times s \mid \sigma \in \Sigma_{s_1 \ldots s_n, s}\}$, like above but without the axioms stating that these predicates have a functional interpretation in models (because the matching logic symbols need not be interpreted as functions).

Translation of Matching Logic (cont.)

We define the translation PL of matching logic (S, Σ)-patterns into predicate logic (S, Π_Σ)-formulae inductively:

$$PL(\varphi) = \forall r \,.\, PL_2(\varphi, r)$$

$$\begin{aligned}
PL_2(x, r) &= (x = r) \\
PL_2(\sigma(\varphi_1, \ldots, \varphi_n), r) &= \exists r_1 \cdots \exists r_n \,.\, PL_2(\varphi_1, r_1) \wedge \cdots \wedge PL_2(\varphi_n, r_n) \wedge \pi_\sigma(r_1, \ldots, r_n, r) \\
PL_2(\neg\varphi, r) &= \neg PL_2(\varphi, r) \\
PL_2(\varphi_1 \wedge \varphi_2, r) &= PL_2(\varphi_1, r) \wedge PL_2(\varphi_2, r) \\
PL_2(\exists x \,.\, \varphi, r) &= \exists x \,.\, PL_2(\varphi, r)
\end{aligned}$$

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

Translation of Matching Logic (cont.)

The predicate logic formula $PL_2(\varphi, r)$ captures the intuition that "r matches φ". The top transformation above, $PL(\varphi) = \forall r \,.\, PL_2(\varphi, r)$, is different from (and simpler than) the corresponding translation of predicates from FOL to predicate logic. It captures the intuition that the pattern φ is valid iff it is matched by *all* values r. Then the following result holds:

Proposition 10.1. *If F is a set of patterns and φ is a pattern, $F \models \varphi$ 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 $\rho[a/r] \models^{=}_{PL} PL_2(\varphi, r)$ for any $\rho : Var \to M$, which follows easily by structural induction on φ. □

Sound and Complete Deduction

- On slides 54 & 55, we propose a sound and complete proof system for matching logic (for total interpretations). The first group (slide 54) of rules/axioms are those of FOL with equality, the second group slide 55) of rules/axioms are about membership.
- We have made no effort to minimize the number of rules and axioms in our proof system. On the contrary, our approach was to include all the rules and axioms that turned out to be useful in proof derivations, especially if they already existed in FOL. Moreover, we preferred to frame unexpected properties of matching logic as axioms or proof rules, so that users of the proof system are fully aware of them.

Theorem 11.2. *The proof systen* in slides 54-55 *is sound and complete:* $F \models \varphi$ *iff* $F \vdash \varphi$.

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

5'. Functional Variable: $\vdash \exists y.\, x = y$

6. Equality Introduction: $\vdash \varphi = \varphi$
7. Equality Elimination: $\vdash \varphi_1 = \varphi_2 \wedge \varphi[\varphi_1/x] \to \varphi[\varphi_2/x]$

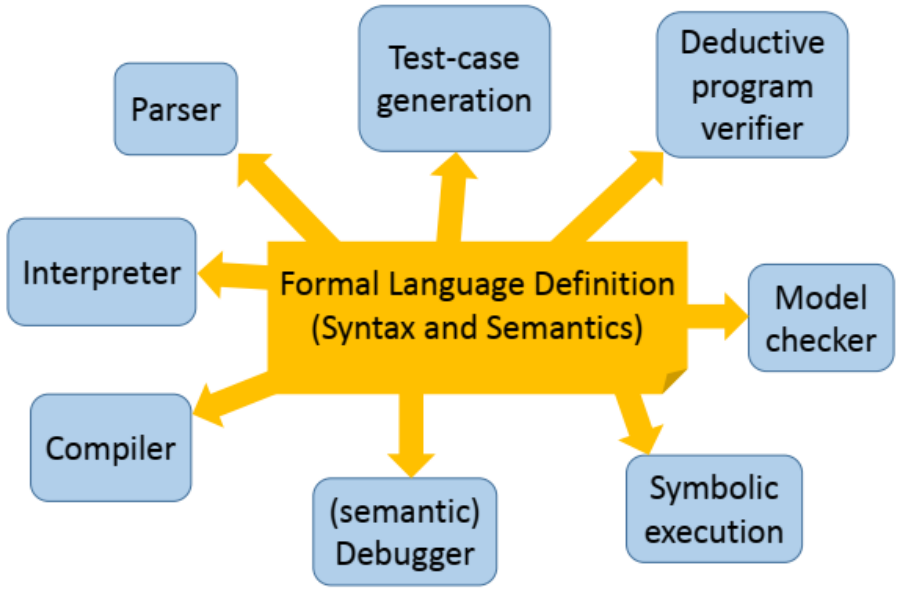
Membership axioms and rules

8. $\vdash \forall x \,.\, x \in \varphi$ iff $\vdash \varphi$
9. $\vdash x \in y = (x = y)$ when $x, y \in Var$
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. $\vdash x \in \sigma(\varphi_1,..,\varphi_{i-1},\varphi_i,\varphi_{i+1},..,\varphi_n) = \exists y.(y \in \varphi_i \wedge x \in \sigma(\varphi_1,..,\varphi_{i-1},y,\varphi_{i+1},..,\varphi_n))$

What to read/watch next

November 1, 2022

- Grigore Rosu, Traian Florin Serbanuta: An overview of the K semantic framework. The Journal of Logic and Algebraic Programming 79 (2010) 397–434.
- Xiaohong Chen, Dorel Lucanu, Grigore Rosu: Matching logic explained. Journal of Logical and Algebraic Methods in Programming 120 (2021) 100638
- Grigore Rosu: Matching Logic: Foundation of the K Framework. Presented at CPP'20, co-located at POPL 2020. https://www.youtube.com/watch?v=Awsv0BlJgbo
- Васил Дядов: Формальная операционная семантика в промышленной разработке (на примере K-framework). Presentation for ruSTEP, 2022. https://youtu.be/2Zz4eInn3AM.



Architecture of the $\mathbb{K}$ framework, powered by matching logic