Matching Logic

Grigore Rosu

Everett Hildenbrandt

Motivation

Separation Logic[1]

Just Add Logic!

Separation logic allows reasoning about small "heaplets" at a time, which simplifies the formulae for proving properties of programs. It does this by introducing several things into the logic itself, listed here:

- Theories of integers and booleans
- Memory maps - $E \mapsto F$
- Predicates $\texttt{isatom?}(E)$ and $\texttt{isloc?}(E)$
- Spatial connectives $\texttt{emp}$, $P * Q$, $P \rightsquigarrow Q$

Matching Logic

- Separation logic expressible without extra primitives in matching logic.
- Reasoning about locations, values, and storage left to programming language semantics.

No Extra Logic

Separation logic isn't the only culprit of the "Just Add Logic!" camp. Matching logic allows for reasoning about *anything* structured without adding to matching logic itself. Thus, matching logic's sound and complete proof system can be used for *any* programming language without modification.

Clean Syntactic Reasoning

Matching logic also lends itself well to syntactic execution.[1]

- Patterns specify sets of elements from a model.
- No need to generate the sets of elements; the pattern itself suffices for reasoning and execution.
- Ex: pattern $\exists x.succ(x)$ grabs all the elements that can be generated by the *succ* symbol.

[1] Because we can reason about matching logic using only the syntax of matching logic expressions, we can write down its proof system as a rewriting logic specification. Rewriting logic is a logic of execution.

Examples

Peano Naturals

Signature (S, Σ)

$$\begin{aligned} S &: \{Nat\} \\ \Sigma &: \{0_{Nat}, succ_{Nat,Nat}, plus_{Nat \times Nat, Nat}\} \\ Var &: \{x_{Nat}, y_{Nat}, z_{Nat}\} \end{aligned}$$

Axioms F

In order to specify *exactly* the Peano Naturals, we add axioms F that an admissible model M must satisfy, $M \models F$.

$$\begin{aligned} 0 \text{ total func} &: \exists y. 0 = y \\ succ \text{ total func} &: \exists y. succ(x) = y \\ succ \text{ inj. func} &: succ(x) = succ(y) \to x = y \\ \text{Only } 0 \text{ or } succ &: 0 \vee \exists x. succ(x) \\ plus \text{ defn} &: plus(x, y) = (x = 0 \wedge y \vee \exists z. succ(z) = x \wedge succ(plus(z, y))) \end{aligned}$$

The *matching logic specification* is the triple (S, Σ, F).

Lists and Memory Maps

Signature (S, Σ)

$$
\begin{aligned}
S&: \{Nat, Seq, Map\} \\
\Sigma&: \{\epsilon_{Seq},\quad emp_{Map},\quad \_\cdot\_{}_{Nat \times Seq, Seq}, \\
&\quad \_ \mapsto \_{}_{Nat \times Nat, Map},\quad \_ \mapsto [\_]_{Nat \times Seq, Map} \\
&\quad \_ * \_{}_{Map \times Map, Map},\quad list_{Nat \times Seq, Map}\} \\
Var&: \{a_{Nat}, b_{Nat}, S_{Seq}\} \cup Var_{Peano}
\end{aligned}
$$

Axioms F

$$
\begin{aligned}
\text{0 not key}&: \neg(0 \mapsto a) \\
\text{Unique keys}&: (x \mapsto a * y \mapsto b) \rightarrow x \neq y \\
\text{Empty } Map&: x \mapsto [\epsilon] = emp \\
Map \text{ ``cons''}&: x \mapsto [a \cdot S] = x \mapsto a * succ(x) \mapsto [S] \\
\text{Empty } list&: list(0, \epsilon) = emp \\
list \text{ ``cons''}&: list(x, a \cdot S) = \exists z. x \mapsto [a \cdot z \cdot \epsilon] * list(z, S)
\end{aligned}
$$

Our specification is $(S \cup S_{Peano}, \Sigma \cup \Sigma_{Peano}, F \cup F_{Peano})$.

Definitions

Signatures and Patterns

Signature (S, Σ) and Variables

S: Set of sorts
Σ: Sort-indexed set of symbols $\Sigma = \{\Sigma_{s_1 \ldots s_n, s}\}_{s_1, \ldots, s_n, s \in S}$
Var: Sort-indexed set of symbols $Var = \{Var_s\}_{s \in S}$

Patterns (Formulae) ϕ_s

ϕ_s: $x \in Var_s \mid \sigma(\phi_{s_1}, \ldots, \phi_{s_n}) \mid \neg\phi_s \mid \phi_s \wedge \phi_s \mid \exists y.\phi_s$
with $\sigma(\phi_{s_1}, \ldots, \phi_{s_n}) \in \Sigma_{s_1 \ldots s_n, s}$ and $y \in Var$

- $x : s \equiv x \in Var_s$
- $\bot_s \equiv x : s \wedge \neg x : s$ $\quad\quad \top_s \equiv \neg\bot_s$
- $\vee$, $\rightarrow$, $\leftrightarrow$, and $\forall$ defined in the normal FOL ways.

$PATTERN$: Sort-indexed set of patterns $PATTERN = \{PATTERN_s\}_{s \in S}$.
We can say $\phi_s \in PATTERN_s$.

Models and M-valuations

(S, Σ) Model M

Carrier set M[2]**:** Sort-indexed set $\{M_s\}_{s \in S}$ is *carrier set of sort s in M*.

Functions σ_M: For each $\sigma_{s_1 \dots s_n, s} \in \Sigma$, define a function $\sigma_M : M_{s_1} \times \dots \times M_{s_n} \to \mathcal{P}(M_s)$, the *interperetation of σ in M*. We may smoothly and usefully say that $\sigma_M(A_1, \dots, A_n) = \cup\{\sigma_M(a_1, \dots, a_n) \mid a_1 \in A_1, \dots, a_n \in A_n\}$ with $A_1 \subseteq M_{s_1}, \dots, A_n \subseteq M_{s_n}$.

M-valuation ρ

M-valuation ρ: $\rho : Var \to M$ selects an element $m \in M_s$ for each $x \in Var_s$.

Extension to $\overline{\rho}$: $\overline{\rho} : PATTERN \to \mathcal{P}(M)$ returns elements of M which "match" the given pattern with M-valuation ρ.

[2] With restriction $M_s \neq \emptyset$ for all $s \in S$.

Pattern Matching

We have defined the M-valuation $\rho : Var \to M$, and hinted at its extension $\overline{\rho} : PATTERN \to \mathcal{P}(M)$. Intuitively, $\overline{\rho}$ takes a pattern ϕ_s and returns all elements of M_s which match the pattern given ρ.

Definition of $\overline{\rho}$

$\overline{\rho}$ is defined recursively on the structure of patterns ϕ_s.

- $\overline{\rho}(x) = \{\rho(x)\}$, for $x \in Var$
- $\overline{\rho}(\neg\phi_s) = M_s \setminus \overline{\rho}(\phi_s)$
- $\overline{\rho}(\sigma(\phi_1, ..., \phi_n)) = \sigma_M(\overline{\rho}(\phi_1), ..., \overline{\rho}(\phi_n))$
- $\overline{\rho}(\phi_1 \wedge \phi_2) = \overline{\rho}(\phi_1) \cap \overline{\rho}(\phi_2)$
- $\overline{\rho}(\exists x.\phi) = \cup\{\overline{\rho'}(\phi) \mid \rho' : Var \to M, \rho'\big|_{FV(\phi)\setminus\{x\}} = \rho\big|_{FV(\phi)\setminus\{x\}}\}$

$M_s \setminus \overline{\rho}(\phi_s)$ is *set difference* $\{m \in M_s \mid m \notin \overline{\rho}(\phi_s)\}$.
$\rho\big|_{FV(\phi)}$ is ρ *with domain restricted to free vars of* ϕ.

Satisfaction and Matching Logic Specification

Model Satisfaction $M \models \phi_s$

M satisfies ϕ_s: $M \models \phi_s$ iff $\overline{\rho}(\phi_s) = M_s$ for all $\rho : Var \to M$.

- What does $M \models x$ mean?
- And $M \models \exists x.\sigma(\phi_1, x) \vee \neg\phi_2$?

ϕ_s *valid*: $\models \phi_s$ iff $M \models \phi_s$ for all M.

M satisfies F: $M \models F$ for $F \subseteq PATTERN$ iff $M \models \phi$ for all $\phi \in F$.

F entails ϕ: $F \models \phi$ iff $M \models F$ implies $M \models \phi$.

Matching Logic Specification (S, Σ, F)

A *matching logic specification* (S, Σ, F) is S a set of sorts, Σ a set of sort-indexed symbols, and F a set of patterns.

Pattern Equality

$M \models \phi_1 = \phi_2$

Equality in matching logic acts as a predicate on the patterns being tested. Two patterns ϕ_1 and ϕ_2 should be equal in a model when they always produce the same elements from the model's carrier set, regardless of ρ.

$$\overline{\rho}(\phi_1 = \phi_2) = M \texttt{ if } (\overline{\rho}(\phi_1) = \overline{\rho}(\phi_2) \texttt{ for all } \rho) \texttt{ else } \emptyset$$

What about $M \models \phi_1 \leftrightarrow \phi_2$?

Consider for example the case of natural numbers - what interperatations does the axiom $\exists y.succ(x) \leftrightarrow y$ allow for $succ_M$? It actually allows for $succ$ to be a total, partial, or non-function!

We need the stronger $=$, with the axiom $\exists y.succ(x) = y$, to specify that $succ$ is actually a total function.

Deduction

Proof System

FOL Axioms and Rules

- $\vdash$ propositional tautologies
- Modus ponens: $\vdash \phi_1$ and $\vdash \phi_1 \rightarrow \phi_2$ imply $\vdash \phi_2$
- $\vdash (\forall x.\phi_1 \rightarrow \phi_2) \rightarrow (\phi_1 \rightarrow \forall x.\phi_2)$ when $x \notin FV(\phi_1)$
- Universal generalization: $\vdash \phi$ implies $\vdash \forall x.\phi$
- Substitution: $\vdash (\forall x.\phi) \wedge (\exists y.\phi' = y) \rightarrow \phi[\phi'/x]$
- Equality introduction: $\vdash \phi = \phi$
- Equality elimination: $\vdash \phi_1 = \phi_2 \wedge \phi[\phi_1/x] \rightarrow \phi[\phi_2/x]$

Membership Axioms and Rules

- $\vdash \forall x.x \in \phi$ iff $\vdash \phi$
- $\vdash x \in y = (x = y)$ when $x, y \in Var$
- $\vdash x \in \neg\phi = \neg(x \in \phi)$
- $\vdash x \in \phi_1 \wedge \phi_2 = (x \in \phi_1) \wedge (x \in \phi_2)$
- $\vdash (x \in \exists y.\phi) = \exists y.(x \in \phi)$ with x and y distinct
- $\vdash x \in \sigma(\phi_1, ..., \phi_{i-1}, \phi_i, \phi_{i+1}, ..., \phi_n) = \exists y.(y \in \phi_i \wedge x \in \sigma(\phi_1, ..., \phi_{i-1}, y, \phi_{i+1}, ...\phi_n))$

Conclusion

Matching Logic

Has

- Sound and complete proof system of its own.
- Intuitive reductions to predicate logic and first order logic with equality.
- Good executability properties.

Doesn't Have

- Extra symbols or definitions specific to the programming language (or structure) that you are reasoning about.
- Restrictions on the types of structure you can reason about.

References

Thanks for listening everyone!

[1] P. O'Hearn, J. Reynolds, and H. Yang, "Local reasoning about programs that alter data structures." 2001.

[2] G. Rosu, "Matching Logic - A Logic for Structural Reasoning." 2014.

[3] G. Rosu, "Matching Logic - Extended Abstract." 2015.