Deriving Matching Logic Specifications from Program Annotations

Dorel Lucanu

Alexandru Ioan Cuza University of Iași

EuroProofNet WG3 Meeting

Timișoara, February 8-9, 2023

1. Introduction
2. Matching Logic (ML)
3. Languages/Programs as ML Theories/Patterns
4. From Annotated Programs to ML Patterns and Back
5. Conclusion

Plan

1. Introduction
2. Matching Logic (ML)
3. Languages/Programs as ML Theories/Patterns
4. From Annotated Programs to ML Patterns and Back
5. Conclusion

Example I

```
@assume r == 1 && x == m && y == n
repeat
@invariant y ge 0
@invariant r * x^y == m^n
@modifies r, x, y
{
  if (y % 2 == 0) {
    x = x * x;
    y = y / 2;
  } else {
    r = r * x;
    y = y - 1;
  }
} until (y == 0);
```

Example II

```
@assume r == 1 && x == m && y == n
repeat
@invariant y ge 0
@invariant r * x^y == m^n
@modifies r, x, y
@decreases y
{
  if (y % 2 == 0) {
    x = x * x;
    y = y / 2;
  } else {
    r = r * x;
    y = y - 1;
  }
} until (y == 0);
```

Matching Logic (ML) - a Foundation for K Framework[1]

ML is a minimal logic where

- definition of programming languages and
- behavioral properties of their programs

can uniformly specified.

In ML a program (configuration) is just a term pattern.

Question: What ML pattern corresponds to an annotated program? (representing the properties of the program given by annotations)

[1] https://kframework.org/

In This Talk

- how to extract ML specifications from annotated programs, and, conversely,
- how to transform ML patterns into annotated programs that can be symbolically executed

Plan

1 Introduction

2 Matching Logic (ML)

3 Languages/Programs as ML Theories/Patterns

4 From Annotated Programs to ML Patterns and Back

5 Conclusion

A Brief History of ML

- An alternative to Hoare/Floyd Logic (Roșu, Ellison, Schulte, AMAST 2010)
- (Several Versions of) Reachability Logic (2011-2019)
- (Many-sorted) Matching Logic (Roșu, LMCS 2017)
- Matching mu-Logic (Chen, Roșu, LICS 2019)
- Applicative Matching Logic (Chen, Roșu, TR 2019; Chen, Roșu, Lucanu, JLAMP 2021)
- Polyadic Matching Logic (Chen, Fiedler, 2022)

Syntax

Patterns:

$\varphi ::= x$	elementary variable ($x \in EV$)
$\mid X$	set variabile ($X \in SV$)
$\mid \sigma$	symbol ($\sigma \in \Sigma$)
$\mid \varphi_1\ \varphi_2$	application
$\mid \bot$	bottom
$\mid \varphi_1 \rightarrow \varphi_2$	implication
$\mid \exists x.\varphi$	existential binder
$\mid \mu X.\varphi$ *if* φ *is positive in* X	least fixpoint binder

Semantics Intuitively

M a set
$\rho : EV \cup SV \to M \cup \mathcal{P}(M)$

x	singleton subset $\{\rho(x)\}$
$\mid X$	subset $\rho(X) \subseteq M$
$\mid \sigma$	subset $\sigma_M \subseteq M$
$\mid \varphi_1\ \varphi_2$	$\_ \cdot \_ \colon M \times M \to \mathcal{P}(M)$, extended pointwise to $\_ \cdot \_ \colon \mathcal{P}(M) \times \mathcal{P}(M) \to \mathcal{P}(M)$,
$\mid \bot$	$\emptyset$
$\mid \varphi_1 \to \varphi_2$	$M \setminus (\lvert\varphi_1\rvert_{M,\rho} \setminus \lvert\varphi_2\rvert_{M,\rho})$
$\mid \exists x.\varphi$	$\bigcup_{a \in M} \lvert\varphi\rvert_{M,\rho[a/x]}$
$\mid \mu X.\varphi$	$\mathbf{lfp}(A \mapsto \lvert\varphi\rvert_{M,\rho[A/X]})$

$M \models \varphi$ iff $\lvert\varphi\rvert_{M,\rho} = M$ fora all ρ

Derived Patterns

$$\top \equiv \neg\bot \qquad \varphi_1 \vee \varphi_2 \equiv \neg\varphi_1 \to \varphi_2$$

$$\neg\varphi \equiv \varphi \to \bot \qquad \varphi_1 \wedge \varphi_2 \equiv \neg(\neg\varphi_1 \vee \neg\varphi_2)$$

$$\forall x.\varphi \equiv \neg\exists x.\neg\varphi \qquad \varphi_1 \leftrightarrow \varphi_2 \equiv (\varphi_1 \to \varphi_2) \wedge (\varphi_2 \to \varphi_1)$$

$$\nu X.\,\varphi \equiv \neg\mu X.\,\neg\varphi[\neg X/X] \qquad \varphi \text{ positive in } X$$

Definedness: Language

theory DEF

Symbols: def

Notations: $\lceil \varphi \rceil \equiv \text{def}\ \varphi$

Axioms: (Definedness) $\forall x.\, \lceil x \rceil$

Notations:

$\lfloor \varphi \rfloor \equiv \neg \lceil \neg \varphi \rceil$ `// totality`

$\varphi_1 = \varphi_2 \equiv \lfloor \varphi_1 \leftrightarrow \varphi_2 \rfloor$ `// equality`

$\varphi_1 \subseteq \varphi_2 \equiv \lfloor \varphi_1 \rightarrow \varphi_2 \rfloor$ `// set inclusion`

$x \in \varphi \equiv x \subseteq \varphi$ `// membership`

endtheory

Sorts: Language (partial)[2]

theory SORT **Imports:** DEF

Symbols: $inh, Sort$

Notations:

$\top_s \equiv inh\ \ s$ // inhabitants of sort s

$s_1 \leq s_2 \equiv \top_{s_1} \subseteq \top_{s_2}$ // subsort relation

$\neg_s\varphi \equiv (\neg\varphi) \wedge \top_s$ // negation within sort s

$\forall x{:}s.\, \varphi \equiv \forall x.\, x \in \top_s \to \varphi$ // $\forall$ within sort s

$\exists x{:}s.\, \varphi \equiv \exists x.\, x \in \top_s \wedge \varphi$ // $\exists$ within sort s

$\mu X{:}s.\, \varphi \equiv \mu X.\, X \subseteq \top_s \wedge \varphi$ // μ within sort s

$\nu X{:}s.\, \varphi \equiv \nu X.\, X \subseteq \top_s \wedge \varphi$ // ν within sort s

$\varphi{:}s \equiv \exists z{:}s.\, \varphi = z$ // "typing"

$f : s_1 \otimes \cdots \otimes s_n \ominus\!\!\!\to s \equiv \forall x_1{:}s_1 \ldots \forall x_n{:}s_n.\, \exists y{:}s.\, f\ x_1 \ldots x_n = y$

// functional

Axioms: $s \in \top_{Sort} \leftrightarrow \lceil \top_s \rceil$

$\exists x.\, Sort = x$

$Sort \in \top_{Sort}$

endtheory

[2]Product, sum, and function sorts are defined in Matching Logic Explained paper.

Natural Numbers:

theory NAT
Imports: SORT
Symbols: $Nat, zero, succ$
Axioms:

(Nat Sort)	$Nat \in \top_{Sort}$
(Nat Zero)	$\exists x.\, zero = x$
(Nat Succ)	$succ : Nat \ominus\!\!\to Nat$
(No Conf Succ.Zero)	$succ\ \ zero \neq zero$
(No Conf Succ)	$\forall x{:}Nat.\, \forall y{:}Nat.\, succ\ \ x = succ\ \ y \to x = y$
(Nat Domain)	$\top_{Nat} = \mu D.\, zero \vee succ\ \ D$

endtheory

For any model M of NAT,
$|\top_{Nat}|_M \approx \mathbb{N} = \{0, 1, 2, \ldots\}$,
$|zero|_M \approx \{0\}$,
$|succ\ \ x|_M \approx \{n+1\}$ if $|x|_M \approx \{n\}$
i.e., it is isomorphic with the term $(\{Nat\}, \{zero, succ\})$-algebra

Plan

1. Introduction
2. Matching Logic (ML)
3. Languages/Programs as ML Theories/Patterns
4. From Annotated Programs to ML Patterns and Back
5. Conclusion

Configurations as Patterns

Concrete configuration:

$\langle \mathtt{x} = \mathtt{x} + 1; \rangle_{code} \, \langle \mathtt{x} \mapsto 7 \rangle_{state}$

ML Pattern:

$\tau \equiv pair\,(code\,(asgn\,\mathtt{x}\,(plus\,\mathtt{x}\,1)))\,(state\,(mapsto\,\mathtt{x}\,7))$

Remark: τ is just the AST of the configuration.

Symbolic Configuration:

$\langle \mathtt{x} = \mathtt{x} + 1; \rangle_{code} \, \langle \mathtt{x} \mapsto \$x \rangle_{state} \, \langle \mathtt{x} > 3 \rangle_{pc}$

ML Pattern:

$\varphi \equiv pair\,(code\,(asgn\,\mathtt{x}\,(plus\,\mathtt{x}\,1)))\,(state\,(mapsto\,\mathtt{x}\,\$x)) \wedge \$x > 3 = true$

The semantics of a symbolic configuration patterns is a set of concrete patterns:

$\tau \in \exists \$x : Nat.\, \varphi$

Transition Systems $(Cfg, \Rightarrow)$ in ML

theory NEXT

Imports: ...

Symbols: $\bullet$

Notations:

(All Path) $\circ X = \neg \bullet \neg X$

Axioms:

(One Path I) $\bullet X \subseteq \top_{Cfg}$

(One Path II) $\lceil \bullet X \rceil \to (X \subseteq \top_{Cfg})$

endtheory

Intended meaning:

$\bullet \varphi'$	$\{\tau \mid \exists \tau'. \tau \Rightarrow \tau' \wedge \tau' \in \varphi'\}$	at least one next config. in φ'
$\circ \varphi'$	$\{\tau \mid \forall \tau'. \tau \Rightarrow \tau' \to \tau' \in \varphi'\}$	all next config. in φ'
$=$		
$\neg \bullet \neg \varphi'$	$\overline{\{\tau \mid \exists \tau'. \tau \Rightarrow \tau' \wedge \neg(\tau' \in \varphi')\}}$	

Particular Cases

$\bullet\bot$	$\emptyset$	
$\circ\bot$	$\{\tau \mid \neg(\exists\tau.\, \tau \Rightarrow \tau')\}$	irreducible config.
$\bullet\top$	$\{\tau \mid \exists\tau.\, \tau \Rightarrow \tau'\}$	reducible config.
$\circ\top$	$\top_{Cfg}$	

Semantic Rules as ML Patterns

$$\langle \texttt{if}\ (e)\ s_1\ \texttt{else}\ s_2 \rightsquigarrow \kappa\rangle_{code}\ \langle\sigma\rangle_{state} \wedge \neg e : Value \to \bullet\langle e \rightsquigarrow \texttt{if}\ (\_)\ s_1\ \texttt{else}\ s_2 \rightsquigarrow \kappa\rangle_{code}\ \langle\sigma\rangle_{state}$$

$$\langle v \rightsquigarrow \texttt{if}\ (\_)\ s_1\ \texttt{else}\ s_2 \rightsquigarrow \kappa\rangle_{code}\ \langle\sigma\rangle_{state} \wedge v : Value \to \bullet\langle \texttt{if}\ (v)\ s_1\ \texttt{else}\ s_2 \rightsquigarrow \kappa\rangle_{code}\ \langle\sigma\rangle_{state}$$

$$\begin{aligned}\langle \texttt{if}\ (b)\ s_1\ \texttt{else}\ s_2 \rightsquigarrow \kappa\rangle_{code}\ \langle\sigma\rangle_{state} \wedge b : \texttt{bool} \to\ & \bullet\langle s_1 \rightsquigarrow \kappa\rangle_{code}\ \langle\sigma\rangle_{state} \wedge b = \texttt{true} \\ & \vee \\ & \bullet\langle s_2 \rightsquigarrow \kappa\rangle_{code}\ \langle\sigma\rangle_{state} \wedge b = \texttt{false}\end{aligned}$$

$$\langle \texttt{@assume}\ \psi; \rightsquigarrow \kappa\rangle_{code}\ \langle\sigma\rangle_{state} \to \bullet\langle\kappa\rangle_{code}\ \langle\sigma\rangle_{state} \wedge \psi$$

$$\langle \texttt{@havoc}\ xs; \rightsquigarrow \kappa\rangle_{code}\ \langle\sigma\rangle_{state} \to \bullet\langle\kappa\rangle_{code}\ \langle\sigma[vs/xs]\rangle_{state}$$

$$\langle \texttt{@assert}\ \psi; \rightsquigarrow \kappa\rangle_{code}\ \langle\sigma\rangle_{state} \wedge \psi \to \bullet\langle\kappa\rangle_{code}\ \langle\sigma\rangle_{state}$$

where *xs* is a list of program variables, *vs* is a list of fresh symbolic variables

ML Patterns Specifying Executions

all-path finally	$\Box\,\psi \equiv \mu X.\, \psi \vee (\circ X \wedge \bullet\top)$
weak all-path finally	$\Box_w\,\psi \equiv \nu X.\, \psi \vee (\circ X \wedge \bullet\top)$

Since $\circ\bot \wedge \bullet\top = \neg\bullet\neg\bot \wedge \bullet\top = \neg\bullet\top \wedge \bullet\top = \bot$ and $\psi \vee (\circ\bot \wedge \bullet\top) = \psi$,

$\Box\psi = \psi \vee (\circ\psi \wedge \bullet\top) \vee (\circ(\circ\psi \wedge \bullet\top) \wedge \bullet\top) \vee \cdots$

Since $\psi \vee (\circ\top \wedge \bullet\top) = \psi \vee \bullet\top$,

$\Box_w\psi = ((\psi\vee\bullet\top)\wedge(\psi\vee(\circ(\psi\vee\bullet\top)\wedge\bullet\top))\wedge(\psi\vee(\circ((\psi\vee(\circ(\psi\vee\bullet\top)\wedge\bullet\top))\wedge\bullet\top))\wedge\cdots$

Plan

1 Introduction

2 Matching Logic (ML)

3 Languages/Programs as ML Theories/Patterns

4 From Annotated Programs to ML Patterns and Back

5 Conclusion

Deriving Reachability Patterns from Annotations

Given

```
repeat
  @modifies bxs
  @invariant ψ
  body
until(E)
```

we derive an ML pattern for the invariant (a new proof obligation)

$$\forall bvs.\ \langle body \rangle_{code}\, \langle \sigma[bvs/bxs] \rangle_{state} \wedge \psi \wedge (bvs = bxs) \rightarrow$$
$$\Box_w\, \exists \sigma'.\ \langle . \rangle_{code}\, \langle \sigma' \rangle_{state}\, \langle \psi \rangle_{pc}$$

and a semantic rule for the annotated statement:

$$\left\langle \begin{array}{l} \texttt{repeat} \\ \quad \texttt{@invariant}\ \psi \\ \quad \texttt{@modifies}\ bxs \\ \quad S \rightsquigarrow \kappa \\ \texttt{until}(E) \end{array} \right\rangle_{code} \langle \sigma \rangle_{state} \rightarrow \bullet \left\langle \begin{array}{l} \texttt{@assert}\ \psi; \\ \texttt{@havoc}\ bxs; \\ \texttt{@assume}\ \psi \wedge E; \rightsquigarrow \kappa \end{array} \right\rangle_{code} \langle \sigma \rangle_{state}$$

Back From ML Patterns to Symbolic Execution

The pattern

$$\forall vs.\ \langle code \rangle_{code} \langle \sigma[vs/xs] \rangle_{state} \wedge \phi \rightarrow$$
$$\Box\, \exists \sigma'.\ \langle . \rangle_{code} \left\langle \sigma' \right\rangle_{state} \wedge \psi$$

corresponds to

$$\left\langle \begin{array}{l} \texttt{@havoc}\ xs; \\ \texttt{@assume}\ \phi; \\ code \\ \texttt{@assert}\ \psi; \end{array} \right\rangle_{code} \quad \langle \cdot \rangle_{state} \Rightarrow^{\forall} \langle \cdot \rangle_{code} \langle \_ \rangle_{state}$$

Plan

1 Introduction

2 Matching Logic (ML)

3 Languages/Programs as ML Theories/Patterns

4 From Annotated Programs to ML Patterns and Back

5 Conclusion

Concluding remarks

We presented

- how annotated programs can be transformed into proof obligations, expressed as ML patterns, and
- how the symbolic execution can be used to check ML patterns expressing program properties

The approach was successfully applied in Alk Platform (https://github.com/alk-language/java-semantics)

References

Xiaohong Chen, Dorel Lucanu, Grigore Rosu: Matching logic explained. J. Log. Algebraic Methods Program. 120: 100638 (2021)

Lungu Alexandru-Ioan, Dorel Lucanu: Supporting Algorithm Analysis with Symbolic Execution in Alk. TASE 2022: 406-423

Lungu Alexandru-Ioan, Dorel Lucanu: A Matching Logic Foundation for Alk. ICTAC 2022: 290-304

Questions?

Thanks!