Matching Logic
An Alternative to Hoare/Floyd Logic

Grigore Rosu
University of Illinois at Urbana-Champaign (UIUC)
Joint work with
Chucky Ellison (UIUC)
Wolfram Schulte (Microsoft Research)
How It Started

• NASA project runtime verification effort
  – Use runtime verification guarantees to ease the task for program verification

• Thus, looked for “off-the-shelf” verifiers
  – Very disappointing experience …
Our Little Benchmark

Reversing of a C list: if \( x \) points to a lists at the beginning, then \( p \) points to its reverse at the end

\[
p = 0; \\
\text{while}(x \neq 0) \{ \\
\quad y = *(x + 1); \\
\quad *(x + 1) = p; \\
\quad p = x; \\
\quad x = y; \\
\}
\]

We were willing to even annotate the program.
Current State of the Art

• Current program verifiers are based on Hoare logic (and WP), separation logic, dynamic logic

• Hoare-logic-based
  – Caduceus/Why, VCC, HAVOC, ESC/Java, Spec#
  – Hard to reason about heaps, frame inference difficult; either (very) interactive, or very slow, or unsound

• Separation-logic-based
  – Smallfoot, Bigfoot, Holfoot* (could prove it! 1.5s), jStar
  – Very limited (only memory safety) and focused on the heap; Holfoot, the most general, is very slow
Current State of the Art

... therefore, we asked for professional help: Wolfram Schulte (Spec# and other tools)
Do we Have a Problem in what regards Program Verification?

• Blame is often on tools, such as SAT/SMT solvers, abstractions, debuggers, static analyzers, slow computers, etc.,
• ... but not on the theory itself, Hoare/Floyd logic – and its various extensions
• Do we need a fresh start, a different way to look at the problem of program verification?
Overview

- Hoare/Floyd logic
- Matching Logic
- Short Demo
- Relationship between Matching Logic and Hoare Logics
- Conclusion and Future Work
Hoare/Floyd Logic

• Assignment rules
  – Hoare (backwards, but no quantifiers introduced)
    
    \[ \{\varphi[e/x]\} \ x := e \ \{\varphi\} \]

  – Floyd (forwards, but introduces quantifiers)
    
    \[ \{\varphi\} \ x := e \ \{\exists v. (x = e[v/x]) \land \varphi[v/x]\} \]
Hoare/Floyd Logic

• Loop invariants

\[
\{\varphi \land (e \neq 0)\} \text{ s } \{\varphi\} \\
\{\varphi\} \text{ while } (e) \text{ s } \{\varphi \land (e = 0)\}
\]

• Minor problem: does not work when \(e\) has side effects; those must be first isolated out
Hoare/Floyd Logic

Important observation

Hoare/Floyd logic, as well as many other logics for program verification, deliberately stay away from “low-level” operational details, such as program configurations

... missed opportunity
What We Want

• Forwards
  – more intuitive as it closely relates to how the program is executed; easier to debug; easier to combine with other approaches (model checking)

• No quantifiers introduced

• Conventional logics for specifications, say FOL

• To deal at least with existing languages and language extensions
  – E.g., Hoare logic has difficulty with the heap; separation logic only deals with heap extensions
Overview

• Hoare/Floyd logic
• Matching Logic
• Short Demo
• Relationship between Matching Logic and Hoare Logics
• Conclusion and Future Work
Matching Logic

• Inspired from operational semantics
  – Program configurations play an important role
• Specifications: special FOL\textsubscript{=} formulae, patterns
• Configurations match patterns
• Patterns can be used to
  1. Give an axiomatic semantics to a language, so that we can reason about programs
  2. Define and reason about patterns of interest in program configurations
Program Configurations (no heap)

• Simple configuration using a computation and an environment

\[
\langle\langle\cdots\rangle_k \langle\cdots\rangle_{env}\rangle
\]

• Example

\[
\langle\langle x := 1; y := 2\rangle_k \langle x \mapsto 3, y \mapsto 3, z \mapsto 5\rangle_{env}\rangle
\]
Program Configurations (add heap)

- Add a heap to the configuration structure:

\[
\langle \langle \cdots \rangle_k \langle \cdots \rangle_{env} \langle \cdots \rangle_{mem} \rangle
\]

- Example

\[
\langle [x] := 5 ; z := [y] \rangle_k \langle x \mapsto 2, y \mapsto 2 \rangle_{env} \langle 2 \mapsto 7 \rangle_{mem}
\]
Patterns

• Configuration terms with constrained variables

\[
\langle\langle x := 1; y := 2 \rangle_k \langle x \mapsto ?a, y \mapsto ?a, ?p\rangle_{\text{env}} \langle ?a \geq 0 \rangle_{\text{form}}\rangle
\]

configuration term with variables constraints

\[
\langle\langle [x] := 5; z := [y] \rangle_k \langle x \mapsto ?a, y \mapsto ?a, ?p\rangle_{\text{env}} \langle ?a \mapsto ?v, ?\sigma\rangle_{\text{mem}} \langle ?a \geq 0 \rangle_{\text{form}}\rangle
\]

configuration term with variables constraints
Pattern Matching

- Configurations match (\( \models \)) patterns iff they match the structure and satisfy the constraints

\[
\langle \langle x := 1; y := 2 \rangle_k \langle x \mapsto 3, y \mapsto 3, z \mapsto 5 \rangle_{env} \rangle \models \\
\langle \langle x := 1; y := 2 \rangle_k \langle x \mapsto \?a, y \mapsto \?a, \?\rho \rangle_{env} \langle \?a \geq 0 \rangle_{form} \rangle
\]

\[
\langle \langle [x] := 5; z := [y] \rangle_k \langle x \mapsto 2, y \mapsto 2 \rangle_{env} \langle 2 \mapsto 7 \rangle_{mem} \rangle \models \\
\langle \langle [x] := 5; z := [y] \rangle_k \langle x \mapsto \?a, y \mapsto \?a, \?\rho \rangle_{env} \langle \?a \mapsto \?v, \?\sigma \rangle_{mem} \langle \?a \geq 0 \rangle_{form} \rangle
\]
What Can We Do With Patterns?

1. Give axiomatic semantics to programming languages, to reason about programs
   – Like Hoare logic, but different

2. Give axioms over configurations, to help identify patterns of interest in them
   – Like lists, trees, graphs, etc.
1. Axiomatic Semantics
- follow the operational semantics -

• Partial correctness pairs:

\[
\frac{\langle \langle e \rangle_k, C \rangle \Downarrow \langle \langle v \rangle_k, C' \rangle}{\langle \langle x = e \rangle_k, C \rangle \Downarrow \langle \cdot \rangle_k, C'[x \leftarrow v] \rangle}
\] (ML-ASGN)

• Assignment

• While

\[
\frac{\langle \langle e \rangle_k, C \rangle \Downarrow \langle \langle v \rangle_k, C' \rangle}{\langle \langle s \rangle_k, (C' \land (v \neq 0)) \rangle \Downarrow \langle \cdot \rangle_k, C \rangle}
\frac{\langle \langle \text{while} (e) s \rangle_k, C \rangle \Downarrow \langle \cdot \rangle_k, (C' \land (v = 0)) \rangle}{(ML\text{-while})}
\]
2. Configuration Axioms

- For example, lists in the heap:

\[ \langle \langle \text{list}(p, \alpha), \sigma \rangle_{\text{mem}} \langle \varphi \rangle_{\text{form}} C \rangle \]

\[ \iff \langle \langle \sigma \rangle_{\text{mem}} \langle p = 0 \land \alpha = \epsilon \land \varphi \rangle_{\text{form}} C \rangle \]

\[ \lor \langle \langle p \mapsto [?a,?q], \text{list}(?q, ?\beta), \sigma \rangle_{\text{mem}} \langle \alpha = ?a: ?\beta \land \varphi \rangle_{\text{form}} C \rangle \]

- Sample configuration properties:

\[ \langle \langle 5 \mapsto 2, 6 \mapsto 0, 8 \mapsto 3, 9 \mapsto 5, \sigma \rangle_{\text{mem}} C \rangle \Rightarrow \langle \langle \text{list}(8, 3 : 2), \sigma \rangle_{\text{mem}} C \rangle , \text{ and} \]

\[ \langle \langle \text{list}(8, 3 : 2), \sigma \rangle_{\text{mem}} C \rangle \Rightarrow \langle \langle 8 \mapsto 3, 9 \mapsto ?q, ?q \mapsto 2, ?q + 1 \mapsto 0, \sigma \rangle_{\text{mem}} C \rangle \]
Overview

• Hoare/Floyd logic
• Matching Logic
• Short Demo: http://fsl.cs.uiuc.edu/ml
• Relationship between Matching Logic and Hoare Logics
• Conclusion and Future Work
Overview

• Hoare/Floyd logic
• Matching Logic
• Short Demo
• Relationship between Matching Logic and Hoare Logics
• Conclusion and Future Work
Matching Logic vs. Hoare Logic

• Hoare logic is equivalent to a fragment of matching logic over simple configurations containing only code and an environment:

\[ \langle \cdots \rangle_k \langle \cdots \rangle_{env} \]

• Thus, any proof derived using Hoare logic can be turned into a proof using matching logic. The opposite not necessarily true
Matching Logic vs. Hoare Logic

Idea of the two transformations:

– Take FOL formulae $\varphi$ into configuration fragments

$$\langle \mathbf{x} \rightarrow ?x, \ldots \rangle_{env} \langle \varphi[?x/\mathbf{x}, \ldots] \rangle_{form}$$

– Take configuration fragments

$$\langle \mathbf{x} \rightarrow ?x, \ldots \rangle_{env} \langle \psi \rangle_{form}$$

into FOL formulae

$$\mathbf{x} = ?x \land \ldots \land \psi$$
Overview

- Hoare/Floyd logic
- Matching Logic
- Short Demo
- Relationship between Matching Logic and Hoare Logics
- Conclusion and Future Work
Concluding Remarks

• Matching logic is derived from operational semantics; it builds upon configurations

• Forwards, can be regarded as a formula-transforming approach. Not the only one:
  – Floyd rule also forwards
  – Evolving specifications (Especs: Pavlovic & Smith)
  – Dynamic logic (Key project – Schmitt etal.)

• Distinctive feature: patterns as symbolic constrained configurations. No artificial “logical encodings” of PL-specific structures needed
Current and Future Work

• Formal rewrite semantics of C (almost finished the complete language definition)
• Using it for runtime analysis of memory safety and for model checking
• To be turned into a matching logic program verifier for C
  – First steps already taken: MatchC
  – Can already be used to prove several runtime verified programs correct