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;
while(x != 0) {
        y = *(x + 1);
        *(x + 1) = p;
        p = x;
        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)

$$\frac{\cdot}{\{\varphi[\mathtt{e}/\mathtt{x}]\}\ \mathtt{x} := \mathtt{e}\ \{\varphi\}}$$

  - Floyd (forwards, but introduces quantifiers)

$$\frac{\cdot}{\{\varphi\}\ \mathtt{x} := \mathtt{e}\ \{\exists v.\ (\mathtt{x} = \mathtt{e}[v/\mathtt{x}]) \wedge \varphi[v\ /\ \mathtt{x}]\}}$$

Hoare/Floyd Logic

- Loop invariants

$$\frac{\{\varphi \wedge (\mathrm{e} \neq 0)\}\ \mathtt{s}\ \{\varphi\}}{\{\varphi\}\ \mathtt{while\,(e)\,s}\ \{\varphi \wedge (\mathrm{e} = 0)\}}$$

- 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 $FOL_=$ 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

$$\langle\langle \mathtt{x} := 1; \mathtt{y} := 2 \rangle_k \; \langle \mathtt{x} \mapsto 3,\; \mathtt{y} \mapsto 3,\; \mathtt{z} \mapsto 5 \rangle_{env} \rangle$$

Program Configurations (add heap)

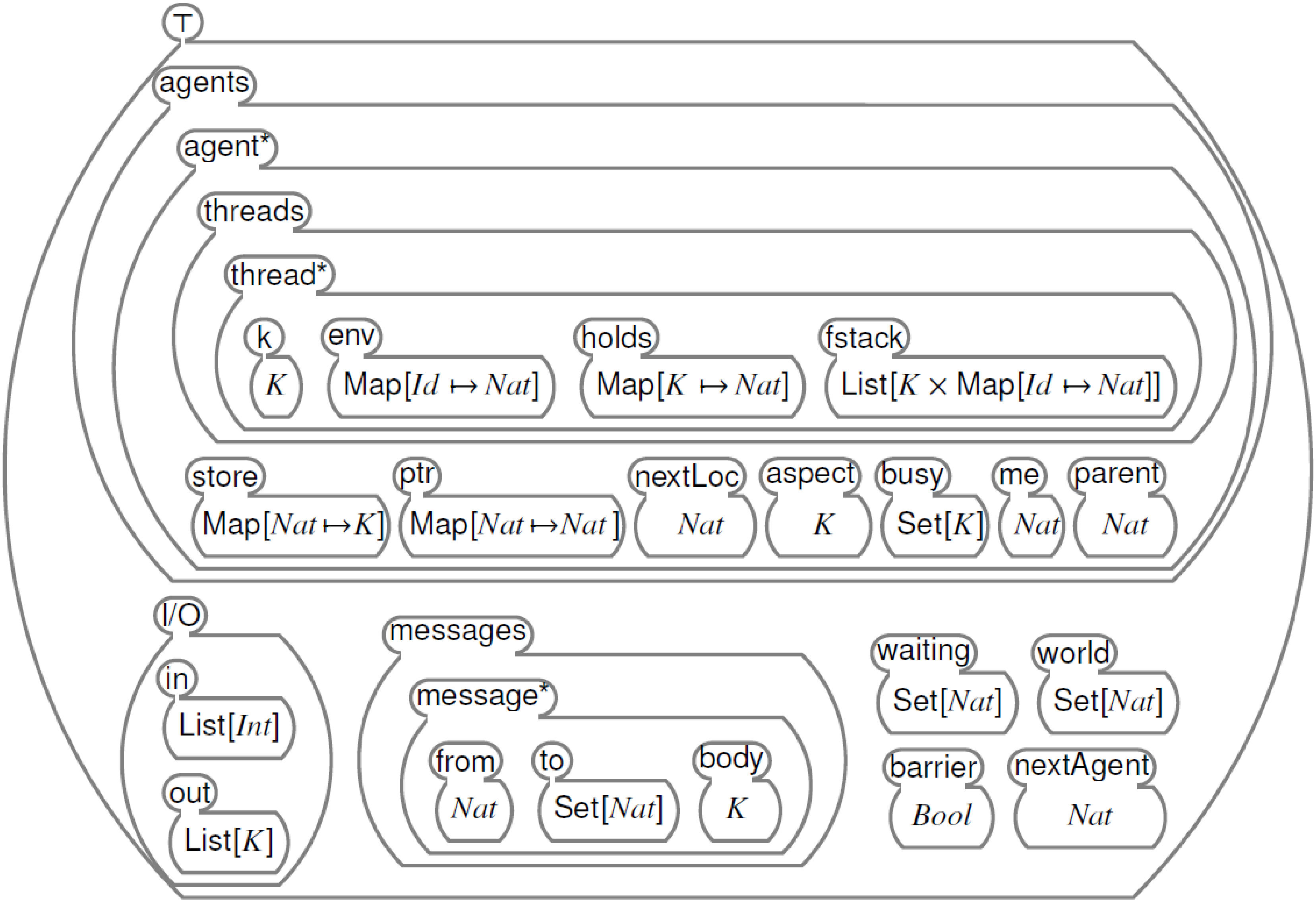
- Add a heap to the configuration structure:

$$\langle \langle \cdots \rangle_k \ \langle \cdots \rangle_{env} \ \langle \cdots \rangle_{mem} \rangle$$

- Example

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

Complex Program Configuration
The CHALLENGE Language (J.LAP 2010)



Patterns

- Configuration terms with constrained variables

$$\underbrace{\langle\langle \mathtt{x} := 1 \,;\, \mathtt{y} := 2\rangle_k \ \langle \mathtt{x} \mapsto ?a,\ \mathtt{y} \mapsto ?a,\ ?\rho\rangle_{env}}_{\text{configuration term with variables}}\ \underbrace{\langle ?a \geq 0\rangle_{form}}_{\text{constraints}}\rangle$$

$$\underbrace{\langle\langle [\mathtt{x}] := 5 \,;\, \mathtt{z} := [\mathtt{y}]\rangle_k \ \langle \mathtt{x} \mapsto ?a,\ \mathtt{y} \mapsto ?a,\ ?\rho\rangle_{env} \ \langle ?a \mapsto ?v,\ ?\sigma\rangle_{mem}}_{\text{configuration term with variables}}\ \underbrace{\langle ?a \geq 0\rangle_{form}}_{\text{constraints}}\rangle$$

Pattern Matching

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

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

$$\langle\langle \texttt{[x] := 5; z := [y]}\rangle_k \; \langle \texttt{x} \mapsto 2,\ \texttt{y} \mapsto 2\rangle_{env} \; \langle 2 \mapsto 7\rangle_{mem}\rangle \qquad \models$$
$$\langle\langle \texttt{[x] := 5; z := [y]}\rangle_k \; \langle \texttt{x} \mapsto ?a,\ \texttt{y} \mapsto ?a,\ ?\rho\rangle_{env} \; \langle ?a \mapsto ?v,\ ?\sigma\rangle_{mem} \; \langle ?a \geq 0\rangle_{form}\rangle$$

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:

$$\Gamma \Downarrow \Gamma'$$

- Assignment

$$\frac{\left\langle \langle \texttt{e} \rangle_k \; C \right\rangle \Downarrow \left\langle \langle \texttt{v} \rangle_k \; C' \right\rangle}{\left\langle \langle \texttt{x = e} \rangle_k \; C \right\rangle \Downarrow \left\langle \langle \cdot \rangle_k \; C'[x \leftarrow v] \right\rangle} \quad \text{(ML-ASGN)}$$

- While

$$\frac{\left\langle \langle \texttt{e} \rangle_k \; C \right\rangle \Downarrow \left\langle \langle \texttt{v} \rangle_k \; C' \right\rangle \qquad \left\langle \langle s \rangle_k \; (C' \wedge (v \neq 0)) \right\rangle \Downarrow \left\langle \langle \cdot \rangle_k \; C \right\rangle}{\left\langle \langle \texttt{while (e) s} \rangle_k \; C \right\rangle \Downarrow \left\langle \langle \cdot \rangle_k \; (C' \wedge (v = 0)) \right\rangle} \quad \text{(ML-WHILE)}$$

2. Configuration Axioms

- For example, lists in the heap:

$$\langle\langle \mathrm{list}(p, \alpha),\ \sigma\rangle_{mem}\ \langle\varphi\rangle_{form}\ C\rangle$$

$$\Leftrightarrow \langle\langle\sigma\rangle_{mem}\ \langle p = 0 \wedge \alpha = \epsilon \wedge \varphi\rangle_{form}\ C\rangle$$

$$\vee\ \langle\langle p \mapsto [?a, ?q],\ \mathrm{list}(?q, ?\beta),\ \sigma\rangle_{mem}\ \langle \alpha = ?a{:}?\beta \wedge \varphi\rangle_{form}\ C\rangle$$

- Sample configuration properties:

$$\langle\langle 5 \mapsto 2,\ 6 \mapsto 0,\ 8 \mapsto 3,\ 9 \mapsto 5,\ \sigma\rangle_{mem}\ C\rangle \Rightarrow \langle\langle \mathrm{list}(8, 3:2),\ \sigma\rangle_{mem}\ C\rangle, \quad \text{and}$$
$$\langle\langle \mathrm{list}(8, 3:2),\ \sigma\rangle_{mem}\ C\rangle \Rightarrow \langle\langle 8 \mapsto 3,\ 9 \mapsto ?q,\ ?q \mapsto 2,\ ?q+1 \mapsto 0,\ \sigma\rangle_{mem}\ C\rangle$$

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:

$$\langle \langle \cdots \rangle_k \; \langle \cdots \rangle_{env} \rangle$$

- 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 φ into configuration fragments

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

– Take configuration fragments

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

into FOL formulae

$$\mathbf{x} \;=\; ?x \wedge \ldots \wedge \psi$$

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