Main Goal

- **Language-independent** program verification framework
- Derive program properties from **operational semantics**

**Questions:**
- Is it possible?
- Is it practical?

**Answers:**
- Sound and complete proof system, so YES, it is possible!
- Efficient automated verifier MatchC, so YES, it is practical!
Overview

- State-of-the-art in Certifiable Verification
- Our Approach
  - Specifying Reachability Properties
  - Reasoning about Reachability
Operational Semantics

• **Easy** to define and understand
  • Can be regarded as formal “implementations”

• Require **little mathematical knowledge**
  • Great introductory topics in PL courses

• **Scale up** well
  • C (>1000 rules), Java, Scheme, Verilog, ..., defined

• **Executable**, so testable
  • C semantics tested against real benchmarks
Operational Semantics

- Sample rule (may require a configuration context)
  
  \[
  \text{while}(E) \ S \Rightarrow \text{if}(E) \ \{S; \ \text{while}(E) \ S\}
  \]

- Define languages only with rules of the form
  
  \[
  l \Rightarrow r \text{ if } b
  \]
  
  - \(l, r\) are configuration terms
  - \(b\) is a Boolean side condition
Unfortunately ...

- Operational semantics considered **inappropriate** for program verification; proofs are **low-level** and **tedious**:
  - Formalization of and working with transition system
  - Typically by induction
    - on the structure of the program
    - on the number of execution steps
    - etc.
Axiomatic Semantics (Hoare Logic)

- Emphasis on program verification
- Programming language captured as a formal proof system deriving Hoare triples

\{ψ\} code \{ψ'\}

precondition

postcondition
Axiomatic Semantics

- Not easy to define and understand, error-prone
- Not executable, hard to test
- Require program transformations, behavior loss

Write $e = 1$ and you’ve got a wrong semantics!
State-of-the-art in Certifiable Verification

- Define an operational semantics: trusted language model
- Define an axiomatic semantics: for verification purposes
- Prove axiomatic semantics sound for operational semantics
- Now we have trusted verification ... BUT
  - Requires two semantics of the same language
  - C operational semantics took more than 2 years!
  - Must be done individually for each language
Overview

- State-of-the-art in Certifiable Verification

Our Approach

- Specifying Reachability Properties
- Reasoning about Reachability
Our Approach

- Underlying belief: **one semantics for each language!**
  - Executable (testable), easy to define and understand
  - Suitable for program verification, “as is”

- **Approach**: **language-independent proof system**
  - Takes operational semantics *unchanged*
  - Derives program properties
  - Both operational semantics rules and program specifications stated as *reachability rules*
Reachability Rules

- Pairs of configuration predicates

Reachability: Any concrete configuration satisfying $\varphi$ and terminating reaches a configuration satisfying $\varphi'$, in the transition system induced by the operational semantics $S$. 

$\varphi \Rightarrow \varphi'$
Overview

- State-of-the-art in Certifiable Verification
- Our Approach
  - Specifying Reachability Properties
  - Reasoning about Reachability
Reachability Rules
- Operational + Axiomatic -

- Operational flavor

\[
\langle \cdots \langle \text{*x} \cdots \rangle_k \langle \cdots x \mapsto L \cdots \rangle_{\text{env}} \langle \cdots L \mapsto V \cdots \rangle_{\text{heap}} \cdots \rangle_{\text{cfg}}
\]

\[
\Rightarrow \langle \cdots \langle V \cdots \rangle_k \langle \cdots x \mapsto L \cdots \rangle_{\text{env}} \langle \cdots L \mapsto V \cdots \rangle_{\text{heap}} \cdots \rangle_{\text{cfg}}
\]

- Axiomatic flavor

\[
\langle \cdots \langle \text{SUM} \cdots \rangle_k \langle \cdots n \mapsto N, \ s \mapsto 0 \cdots \rangle_{\text{env}} \cdots \rangle_{\text{cfg}} \land N \geq 0
\]

\[
\Rightarrow \langle \cdots \langle \cdots \rangle_k \langle \cdots n \mapsto N, \ s \mapsto N(N + 1)/2 \cdots \rangle_{\text{env}} \cdots \rangle_{\text{cfg}} \land N \geq 0
\]
Hoare Triple = Syntactic Sugar

\[ p = \text{NULL}; \]
\[ \text{inv} \langle \cdots \text{list}(p)(?B), \text{list}(x)(?C) \cdots \rangle_{\text{heap}} \wedge A = \text{rev}(?B)@?C \]
\[ \text{while} (x \neq \text{NULL}) \{ \]
\[ \quad y = x->\text{next}; \]
\[ \quad x->\text{next} = p; \]
\[ \quad p = x; \]
\[ \quad x = y; \]
\[ \} \cdots \rangle_{k} \]
\[ \cdots \]
\[ \rangle_{\text{cfg}} \wedge A = \text{rev}(?B)@?C \]
\[ \Rightarrow \]
\[ \langle \cdots \text{x} \mapsto ?x, \ p \mapsto ?p \cdots \rangle_{\text{env}} \]
\[ \langle \cdots \text{list}(?p)(?B), \text{list}(?x)(?C) \cdots \rangle_{\text{heap}} \]
\[ \langle \cdots \rangle_{k} \]
\[ \cdots \]
\[ \rangle_{\text{cfg}} \wedge A = \text{rev}(?B)@?C \wedge ?x = \text{NULL} \]
Matching Logic

- State *static* properties of program configurations
  - Parametric in a model of configurations

- Extends first-order logic with *patterns*
  - Special predicates which are configuration terms
  - Configurations satisfy patterns iff they match them

- C Configurations

```
⟨...⟩struct ⟨...⟩funs ⟨...⟩k ⟨...⟩env ⟨...⟩tenv
⟨...⟩fname ⟨...⟩stack ⟨...⟩heap ⟨...⟩in ⟨...⟩out
⟨...⟩cfg
```

Extra 70 cells
Model of Configurations

- Properties -

- Configuration abstraction (list)
  - “Separation” achieved at term level

\[
\langle\langle\text{list}(p)(A), \ H\rangle_{\text{heap}} \ C\rangle_{\text{cfg}} \\
\leftrightarrow \langle\langle H\rangle_{\text{heap}} \ C\rangle_{\text{cfg}} \land p = \text{NULL} \land A = \cdot \\
\lor \ \exists a, q, B (\langle\langle p \mapsto [a, q], \ \text{list}(q)(B), H\rangle_{\text{heap}} \ C\rangle_{\text{cfg}} \land A = [a]@B)
\]

- Operations (reverse)

\[
\text{rev}(A_1 @ A_2) = \text{rev}(A_2) @ \text{rev}(A_1)
\]
Separation Logic = Matching Logic Instance

- Separation logic: popular logic for heap properties
- Mechanical translation to matching logic (see paper)
  - Configuration: \( \langle \ldots \rangle_{\text{heap}} \)
  - Separation encoded using different sub-terms
- No expressiveness loss from using matching logic
- Matching logic gives “structural separation” anywhere in the configuration, not only in the heap
Operational and Axiomatic Semantics

Rules as Reachability Rules

- Reachability rules generalize
  - Operational semantics rules
  - Hoare triples

- Operational semantics rule $l \Rightarrow r \text{ if } b$ is syntactic sugar for reachability rule $l \land b \Rightarrow r$

- Hoare triple encoded in a reachability rule with the empty code in the right-hand-side (see FM’12)
Overview

- State-of-the-art in Certifiable Verification
- Our Approach
  - Specifying Reachability Properties
  - Reasoning about Reachability
Reasoning about Reachability

- The main result of our paper is a proof system deriving reachability rules from reachability rules:

\[ A \vdash \varphi \Rightarrow \varphi' \]

- Trusted reachability rules (starts with operational semantics)
- Claimed reachability rules
- Target reachability rule
Reachability Proof System

- 8 Rules -

**Axiom:**
\[
\varphi \Rightarrow \varphi' \in \mathcal{A} \\
\mathcal{A} \vdash \varphi \Rightarrow \varphi'
\]

**Logic Framing:**
\[
\mathcal{A} \vdash \varphi \Rightarrow \varphi' \quad \psi \text{ is a (patternless) FOL formula} \\
\mathcal{A} \vdash \varphi \land \psi \Rightarrow \varphi' \land \psi
\]

**Transitivity:**
\[
\mathcal{A} \vdash \varphi_1 \Rightarrow \varphi_2 \\
\mathcal{A} \cup C \vdash \varphi_2 = \varphi_3 \\
\mathcal{A} \vdash \varphi_1 \Rightarrow \varphi_3
\]

**Consequence:**
\[
\models \varphi_1 \Rightarrow \varphi'_1 \\
\mathcal{A} \vdash \varphi'_1 \Rightarrow \varphi'_2 \\
\models \varphi'_2 \Rightarrow \varphi_2 \\
\mathcal{A} \vdash \varphi_1 \Rightarrow \varphi_2
\]

**Reflexivity:**
\[
\mathcal{A} \vdash \varphi \Rightarrow \varphi
\]

**Case Analysis:**
\[
\mathcal{A} \vdash \varphi_1 \Rightarrow \varphi \\
\mathcal{A} \vdash \varphi_2 \Rightarrow \varphi \\
\mathcal{A} \vdash \varphi_1 \lor \varphi_2 \Rightarrow \varphi
\]

**Circularity:**
\[
\mathcal{A} \vdash \cup_{\varphi_1 \Rightarrow \varphi'_1} \varphi \Rightarrow \varphi' \\
\mathcal{A} \vdash \varphi \Rightarrow \varphi'
\]

**Function:**
\[
\mathcal{A} \vdash \varphi' \\
X \cap \text{FreeVars}(\varphi') = \emptyset \\
\mathcal{A} \vdash \exists X \varphi \Rightarrow \varphi'
\]

Symbolic execution (multiple steps)

Symbolic execution (one step)

Code with circular behavior
Circular behaviors

- Circularity and Transitivity proof rules:
  \[ \begin{align*}
  \mathcal{A} & \vdash \text{Cup} \{ \varphi \Rightarrow \varphi' \} \varphi \Rightarrow \varphi' \\
  \mathcal{A} & \vdash \varphi \Rightarrow \varphi'
  \end{align*} \]
  \[ \begin{align*}
  \mathcal{A} & \vdash \varphi_1 \Rightarrow \varphi_2 \\
  \mathcal{A} & \vdash \varphi_2 \Rightarrow \varphi_3 \\
  \mathcal{A} \cup C & \vdash \varphi_2 \Rightarrow \varphi_3 \\
  \mathcal{A} & \vdash \varphi_1 \Rightarrow \varphi_3
  \end{align*} \]

- Hoare logic rule for while loops:
  \[ \begin{align*}
  \mathcal{H} & \vdash \{ \psi \land e \neq 0 \} \mathbf{s} \{ \psi \} \\
  \mathcal{H} & \vdash \{ \psi \} \text{while}(e) \mathbf{s} \{ \psi \land e = 0 \}
  \end{align*} \]
Soundness

**Theorem:** If $S \vdash \varphi \Rightarrow \varphi'$ is derivable by the proof system, then $S \models \varphi \Rightarrow \varphi'$ is semantically valid.
Relative Completeness

**Theorem:** If $S \models \varphi \Rightarrow \varphi'$ is semantically valid, then $S \vdash \varphi \Rightarrow \varphi'$ is derivable by the proof system, with $S$ the operational semantics of a language.

- Relativity
  - Validity oracle for static configuration properties
- Language-independent result, unlike Hoare logics
MatchC

- Proof-of-concept verifier for a C fragment
- Derives program specifications from the operational semantics (in K framework) using the proof system
  - No Hoare/separation logic, no WP, no VC generation
- Automated, user only provide
  - Specifications for recursive functions and loops
List reverse: code + invariant
Implementation

- Heuristics for applying the proof system
  - (forward) symbolic execution

- Matching logic reasoning
  - Maude: efficient structure matching and rearranging
    - matching a list the heap, ...
  - SMTs (CVC3, Z3): simplifying constraints
    - small queries (milliseconds each)
## Preliminary Evaluation

<table>
<thead>
<tr>
<th>Program</th>
<th>Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Buffered read-write</td>
<td>0.15</td>
</tr>
<tr>
<td>Stack inspection</td>
<td>0.24</td>
</tr>
<tr>
<td>Insertion sort</td>
<td></td>
</tr>
<tr>
<td>Merge sort</td>
<td></td>
</tr>
<tr>
<td>Quicksort</td>
<td>1.97</td>
</tr>
<tr>
<td>AVL find</td>
<td>0.15</td>
</tr>
<tr>
<td>AVL insert</td>
<td>43.5</td>
</tr>
<tr>
<td>AVL delete</td>
<td>133.58</td>
</tr>
<tr>
<td>Schorr-Waite (tree)</td>
<td>0.28</td>
</tr>
<tr>
<td>Schorr-Waite (graph)</td>
<td>1.73</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Dozens more programs at</td>
<td></td>
</tr>
<tr>
<td>[matching-logic.org]</td>
<td></td>
</tr>
</tbody>
</table>

Only annotated main functions (insert/delete). Inlined auxiliary functions (balance, rotate, ...).
Conclusions

- Matching logic reachability proof system
  - Sound and (relatively) complete
  - Practical
- MatchC, an automated verifier
  - Expressive
  - Efficient
- Operational semantics based verification is viable!

matching-logic.org