Towards a Unified Theory of Operational and Axiomatic Semantics

Grigore Rosu and Andrei Stefanescu
University of Illinois, USA
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 of IMP
- Sample Rules -

\[
\begin{align*}
\text{if}(i) \; s_1 \; \text{else} \; s_2 \Rightarrow s_1 \quad \text{if } i \neq 0 \\
\text{if}(0) \; s_1 \; \text{else} \; s_2 \Rightarrow s_2 \\
\text{while}(e) \; s \Rightarrow \text{if}(e) \; s; \text{while}(e) \; s \; \text{else} \; \text{skip} \\
\text{proc}() \Rightarrow \text{body} \quad \text{where} \; \text{``proc() body''}
\end{align*}
\]
Operational Semantics of IMP
- Sample Rules -

\[
\begin{align*}
\text{if}(i) \ s_1 \ \text{else} \ s_2 & \Rightarrow s_1 \quad \text{if } i \neq 0 \\
\text{if}(0) \ s_1 \ \text{else} \ s_2 & \Rightarrow s_2 \\
\text{while}(e) \ s & \Rightarrow \text{if}(e) \ s; \ \text{while}(e) \ s \ \text{else} \ \text{skip} \\
\text{proc}() & \Rightarrow \text{body} \quad \text{where “proc() body”}
\end{align*}
\]

May need to be completed “all the way to top”, into rules between configurations:

\[
\langle C, \sigma \rangle[ \ \text{if}(i) \ s_1 \ \text{else} \ s_2 ] \Rightarrow \langle C, \sigma \rangle[ s_1 ] \quad \text{if } i \neq 0
\]
We can operationally define any programming languages only with rewrite rules of the form

\[ l \Rightarrow r \text{ if } b \]

where \( l, r \) are “top-level” configuration terms, and \( b \) is a Boolean side condition.
Unfortunately ...

- Operational semantics considered inappropriate for program reasoning
- Proofs based on operational semantics are low-level and tedious
  - Have to formalize and work with transition system
  - Induction on structure, number of steps, etc.
AXIOMATIC SEMANTICS
(HOARE LOGIC)
Axiomatic Semantics

- Focused on reasoning
- Programming language captured as a formal proof system that allows to derive triples

{ψ} code {ψ’}

Precondition

Postcondition
Axiomatic Semantics

• Not easy to define and understand, error-prone
  – Not executable, hard to test; require program transformations which may lose behaviors, etc.

\[
\begin{align*}
\mathcal{H} & \vdash \{\psi \land e \neq 0\} \ s \ \{\psi\} \\
\mathcal{H} & \vdash \{\psi\} \ while(e) \ s \ \{\psi \land e = 0\} \\
\mathcal{H} & \cup \{\psi\} \ proc() \ \{\psi'\} \vdash \{\psi\} \ body \ \{\psi'\} \\
\mathcal{H} & \vdash \{\psi\} \ proc() \ \{\psi'\}
\end{align*}
\]
State-of-the-art in Certifiable Verification

- Define an operational semantics, which acts as trusted reference model of the language
- Define an axiomatic semantics, for reasoning
- Prove the axiomatic semantics sound for the operational semantics

- Now we have trusted verification ...
- ... but the above needs to be done for each language individually; at best uneconomical
Unified Theory of Programming - (Hoare and Jifeng) -

• Framework where various semantics of the same language coexist, with systematic relationships (e.g., soundness) proved

• Then use one semantics or another ...

• This still requires two or more semantics for the same language (C semantics took >2years)

• Uneconomical, people will not do it
Unified Theory of Programming
- Our Approach -

• Underlying belief
  – A language should have only one semantics, which should be easy, executable, and good for program reasoning. One semantics to rule them all.

• Approach
  – Devise language-independent proof system that takes operational semantics “as is” and derives any reachability property (including Hoare triples).
Matching Logic
(AMAST’10, ICSE’11, ICALP’12, FM’12, OOPSLA’12)

• Logic for reasoning about structure
• Matching logic: extend FOL with patterns
  – Special predicates which are open configuration terms, whose meaning is “can you match me?”

• Examples of patterns:
  \[
  \langle \text{if } i \ s_1 \ s_2, \ \sigma \rangle \land i \neq 0
  \]

  \[
  \exists s \left( \langle s := 0; \ \text{while}(n > 0)(s := s + n; \ n := n - 1), \right.
  \\
  \left. (s \mapsto s, \ n \mapsto n) \rangle \land n \geq_{\text{Int}} 0 \right)
  \]

  \[
  \langle \text{skip}, \ (s \mapsto n \ast_{\text{Int}} (n +_{\text{Int}} 1)/_{\text{Int}} 2, \ n \mapsto 0) \rangle
  \]
Reachability Rule

• Pair of patterns, with meaning “reachability”

\[ \varphi \Rightarrow \varphi' \]

• Reachability rules generalize both operational semantics rules and Hoare triples
Operational Semantics Rules are Reachability Rules

Operational semantics rule

\[ l \Rightarrow r \text{ if } b \]

is syntactic sugar for reachability rule

\[ l \land b \Rightarrow r \]

We can associate a transition system to any set of reachability rules, and define validity; see paper

\[ S \models \varphi \Rightarrow \varphi' \]
Hoare Triples are Reachability Rules

Hoare triple

\{\psi\} \text{code} \{\psi'\}

is syntactic sugar for reachability rule

\[ \exists X_{\text{code}} (\langle \text{code}, \sigma_{X_{\text{code}}} \rangle \land \psi_X) \Rightarrow \exists X_{\text{code}} (\langle \text{skip}, \sigma_{X_{\text{code}}} \rangle \land \psi'_X) \]

... but there are better ways to specify program properties; see the paper
Reasoning about Reachability

• Having generalized the elements of both operational and axiomatic semantics, we now want a proof system for deriving reachability rules from reachability rules:

\[ \mathcal{A} \vdash \varphi \Rightarrow \varphi' \]
Reachability Proof System
- 9 language-independent rules -

### Rules of operational nature

**Reflexivity**:

\[
  \vdash \varphi \Rightarrow \varphi
\]

**Axiom**:

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

**Substitution**:

\[
  \mathcal{A} \vdash \varphi \Rightarrow \varphi' \\
  \theta : \text{Var} \rightarrow \mathcal{T}_\Sigma(\text{Var}) \\
  \mathcal{A} \vdash \theta(\varphi) \Rightarrow \theta(\varphi')
\]

**Transitivity**:

\[
  \mathcal{A} \vdash \varphi_1 \Rightarrow \varphi_2 \\
  \mathcal{A} \vdash \varphi_2 \Rightarrow \varphi_3 \\
  \mathcal{A} \vdash \varphi_1 \Rightarrow \varphi_3
\]

### Rules of deductive nature

**Case Analysis**:

\[
  \mathcal{A} \vdash \varphi_1 \Rightarrow \varphi \\
  \mathcal{A} \vdash \varphi_2 \Rightarrow \varphi \\
  \mathcal{A} \vdash \varphi_1 \lor \varphi_2 \Rightarrow \varphi
\]

**Logic Framing**:

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

**Consequence**:

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

**Abstraction**:

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

**Rule for circular behavior**

**Circularity**:

\[
  \mathcal{A} \vdash \varphi \Rightarrow \varphi'' \\
  \mathcal{A} \cup \{\varphi \Rightarrow \varphi'\} \vdash \varphi'' \Rightarrow \varphi' \\
  \mathcal{A} \vdash \varphi \Rightarrow \varphi'
\]
Rule 1
Reflexivity

\[ \mathcal{A} \vdash \varphi \implies \varphi \]
Rule 2

Axiom

\[ \varphi \Rightarrow \varphi' \in \mathcal{A} \]

\[ \mathcal{A} \vdash \varphi \Rightarrow \varphi' \]
Rule 3
Substitution

\[ \mathcal{A} \vdash \varphi \Rightarrow \varphi' \]
\[ \theta : \text{Var} \rightarrow \mathcal{T}_\Sigma(\text{Var}) \]

\[ \mathcal{A} \vdash \theta(\varphi) \Rightarrow \theta(\varphi') \]
Rule 4
Transitivity

\[
\begin{align*}
\mathcal{A} & \vdash \varphi_1 \Rightarrow \varphi_2 \\
\mathcal{A} & \vdash \varphi_2 \Rightarrow \varphi_3 \\
\hline
\mathcal{A} & \vdash \varphi_1 \Rightarrow \varphi_3
\end{align*}
\]
Rule 5

Case Analysis

\[ \mathcal{A} \vdash \varphi_1 \Rightarrow \varphi \]

\[ \mathcal{A} \vdash \varphi_2 \Rightarrow \varphi \]

\[ \mathcal{A} \vdash \varphi_1 \lor \varphi_2 \Rightarrow \varphi \]
Rule 6
Logic Framing

\[ \mathcal{A} \vdash \varphi \Rightarrow \varphi' \]

\( \psi \) is a (patternless) FOL formula

\[ \mathcal{A} \vdash \varphi \land \psi \Rightarrow \varphi' \land \psi \]
Rule 7

Consequence

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

\[ \frac{\varphi_2 \rightarrow \varphi_2}{\mathcal{A} \vdash \varphi_1 \Rightarrow \varphi_2} \]
Rule 8

Abstraction

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

\[ X \cap \text{FreeVars}(\varphi') = \emptyset \]

\[ A \vdash \exists X \varphi \Rightarrow \varphi' \]
Rule 9

Circularity

\[ \mathcal{A} \vdash \varphi \Rightarrow^{+} \varphi'' \]

\[ \mathcal{A} \cup \{ \varphi \Rightarrow \varphi' \} \vdash \varphi'' \Rightarrow \varphi' \]

\[ \mathcal{A} \vdash \varphi \Rightarrow \varphi' \]
Main Result

Soundness

**Theorem:** If $\vdash \varphi \Rightarrow \varphi'$ derivable with the nine-rule proof system, then $\models \varphi \Rightarrow \varphi'$
Conclusion

• Proof system for reachability
• Works with any operational semantics, as is
• Requires no other semantics of the language
• Unlike Hoare logics, which are language-specific, our proof system is
  – Language-independent (takes language as axioms)
  – Proved sound only once, for all languages
• Has been implemented in MatchC and works
• Can change the way we do program verification