Step-indexed Semantic Model of the Imperative Object Calculus

Catalin Hritcu  Jan Schwinghammer  Gert Smolka

Programming Systems Lab
Saarland University, Saarbrücken

February 2007
The Big Picture
Deduction systems

\[(\text{SEM-SSub}) \quad \frac{\Gamma \models a : \alpha \quad \alpha \subseteq \beta}{\Gamma \models a : \beta}\]

\[(\text{SEM-SRefl}) \quad \alpha \subseteq \alpha \quad (\text{SEM-STRans}) \quad \frac{\alpha \subseteq \tau \quad \tau \subseteq \beta}{\alpha \subseteq \beta}\]

\[(\text{SEM-STop}) \quad \alpha \subseteq \top \quad (\text{SEM-SBot}) \quad \bot \subseteq \alpha\]

\[(\text{SEM-SObj}) \quad \frac{E \subseteq D}{[m_d = \varsigma(x_d)b_d]_{d \in D} \subseteq [m_e = \varsigma(x_e)b_e]_{e \in E}}\]

\[(\text{SEM-SRec}) \quad \forall \alpha, \beta \in \text{Type}. \quad \alpha \subseteq \beta \Rightarrow F(\alpha) \subseteq G(\beta)\]

\[(\text{SEM-SUniv}) \quad \beta \subseteq \alpha \quad \forall \tau \in \text{Type}. \quad \tau \subseteq \beta \Rightarrow F(\tau) \subseteq G(\tau)\]
Deduction systems

• Type systems
  • Property: type safety - efficiently decidable
  • Soundness usually proved syntactically
  • One can also use semantic models! (details soon)

• Program logics (e.g. Hoare logic)
  • Property: correctness - undecidable
  • Soundness proved wrt. semantic model:
    • “Derivability implies validity in the model”
The Challenge
The Challenge

- Executable code on the heap (e.g. C/C++, Java, ML)
- Under this realistic assumption
  - Denotational semantic models
    - Very complex
    - Still not perfect
The Challenge

- Executable code on the heap (e.g. C/C++, Java, ML)
- Under this realistic assumption
  - Denotational semantic models
    - Very complex
    - Still not perfect
- Possible alternative: “step-indexing”
  - Simpler set-theoretic models (details soon)
Imperative Object Calculus

- Simple object-oriented programming language
- Only one primitive: objects
- Classes and inheritance encoded
- Captures the essence of object-oriented languages
- Features dynamically allocated, higher-order store
- We also studied the functional object calculus
- A Theory of Objects
  [Abadi & Cardelli ‘96]

Martin Abadi  Luca Cardelli
Imperative Object Calculus

\[ a, b ::= x \quad \text{variable} \]
\[ \quad [m_d = \varsigma(x_d)b_d]_{d \in D} \quad \text{object creation} \]
\[ \quad \text{clone } a \quad \text{object cloning} \]
\[ \quad a.m \quad \text{method invocation} \]
\[ \quad a.m ::= \varsigma(x)b \quad \text{method update} \]
\[ \quad \lambda x. b \mid a b \quad \text{procedures} \]

\[ [gcd = \varsigma(y)\lambda x. \lambda z. \text{if } x < z \text{ then } y.gcd \, x \, (z - x) \]
\[ \quad \text{else if } z < x \text{ then } y.gcd \, (x - z) \, z \text{ else } x] \]
Step-indexed Semantic Models

- First developed by Appel et al. - foundational PCC
- Machine-checkable proofs of type soundness
  - For low-level languages
- Lambda calculus
  - Recursive types [Appel & McAllester, ‘01]
- General references and polymorphism
  [Ahmed et al., ‘03] [Ahmed, ‘04]
Step-indexed Semantic Models

- Use operational semantics of untyped calculus
- Types are sets of indexed values
  \[ a : k \alpha \text{ if } a \text{ behaves like an element of } \alpha \text{ for } k \text{ steps} \]
- In the presence of state
  - Values also indexed by store typings \( \Psi \)
  - Store extension relation: \( (k, \Psi) \sqsubseteq (j, \Psi') \)
- Types have a “meaning” - predicates over programs
Types

- **Object types** $[m_d : \tau_d]_{d \in D}$
- **Recursive types** $\mu F$
- **Polymorphic types** $\forall F \quad \exists F$
- **Subtyping** $\alpha \subseteq \beta$

- "No object calculus can fully justify its existence without some notion of subsumption" - Luca Cardelli

- **Bounded quantification** $\forall_\alpha F \quad \exists_\alpha F$

- **Self types ~ recursive object types**
  \[ \varsigma(\lambda \tau \in \text{Type}. \ [m_d : F_d(\tau)]_{d \in D}) \]
Rules and Soundness

- 27 semantic typing rules (+31 functional obj. calc.)

\[(\text{SEM-PACK}) \quad \exists \tau \in \text{Type}. \, \tau \subseteq \alpha \land \Gamma \models a : F(\tau) \quad \Gamma \models \text{pack} \, a : \exists \alpha F\]

\[(\text{SYN-PACK}) \quad \Psi, \Gamma \vdash C \leq A \quad \Psi, \Gamma \vdash a[X := C] : B[X := C] \quad \Psi, \Gamma \vdash \text{pack} \, X \leq A = C \, a : \exists X \leq A. \, B\]

- Note: we only check location-free programs

- Soundness proofs
  - Foundational, i.e. elementary
  - Easily machine checkable (not checked yet)
Further Work

- Local and modular programming logic
- ... for the imperative object calculus
- Proving it sound wrt. semantic model
Program Logics

Hoare Logic (1967, 1969)

Robert Floyd

Tony Hoare
not local


local

modular

John Reynolds

Program Logics

not modular

Hoare Logic (1967, 1969)
Program Logics

<table>
<thead>
<tr>
<th>Local</th>
<th>Not Local</th>
</tr>
</thead>
<tbody>
<tr>
<td>Modular</td>
<td>Not Modular</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Not Modular</th>
<th>Modular</th>
</tr>
</thead>
</table>

Peter O'Hearn  
John Reynolds
Program Logics

<table>
<thead>
<tr>
<th>not local</th>
<th>local</th>
</tr>
</thead>
<tbody>
<tr>
<td>modular</td>
<td></td>
</tr>
<tr>
<td>Hoare Logic (1967, 1969)</td>
<td></td>
</tr>
<tr>
<td>not modular</td>
<td></td>
</tr>
</tbody>
</table>

Bernhard Reus  Jan Schwinghammer
Program Logics

not local


local

Sep. and Abstraction (2005)

modular

Hoare Logic (1967, 1969)

not modular

Separation Logic (2001)

Higher-order Store (2006)

Matthew Parkinson
Program Logics

<table>
<thead>
<tr>
<th>not local</th>
<th>local</th>
</tr>
</thead>
<tbody>
<tr>
<td>modular</td>
<td></td>
</tr>
<tr>
<td>not modular</td>
<td></td>
</tr>
</tbody>
</table>

Aleks Nanevski
Amal Ahmed
Greg Morrisett
Lars Birkedal
<table>
<thead>
<tr>
<th>not local</th>
<th>local</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>modular</th>
</tr>
</thead>
<tbody>
<tr>
<td>Neel. Krishnaswami</td>
</tr>
<tr>
<td>Lars Birkedal</td>
</tr>
<tr>
<td>Jonathan Aldrich</td>
</tr>
<tr>
<td>John Reynolds</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>not modular</th>
</tr>
</thead>
<tbody>
<tr>
<td>Higher-order Store (2006)</td>
</tr>
</tbody>
</table>
not local


modular

Hoare Logic (1967, 1969)

not modular


local

Sep. and Abstraction (2005)

Hoare Type Theory (2007)

Separation Logic (2001)


Object Calculus

Martin Abadi

Rustan Leino

Higher-order Store (2006)

Sep. and Abstraction (2005)

Hoare Logic (1967, 1969)

Separation Logic (2001)

Higher-order Store (2006)

Hoare Type Theory (2007)


Jan Schwinghammer
Summary

• Imperative object calculus
  • Step-indexed semantic model of types
    • Main contributions
      • Object types
      • Subtyping
        • Bounded quantification, self types
    • One direction for further research
      • Local and modular program logic