International Max Planck Research School
for Computer Science

Semantics of Imperative Objects

Catalin Hritcu

Supervisor: *Jan Schwinghammer*
Responsible: *Prof. Gert Smolka*
Programming Systems Lab
Saarland University, Saarbrücken

November 2006

Programming Languages

- Syntax (how programs look like)
  - Context-free grammars
  - Abstract syntax trees
- Semantics (the meaning of programs)
  - Operational, denotational, axiomatic semantics
  - Type systems
  - Step-indexed semantics

Object Calculi

- Object-oriented programming languages
- Idealized models
- Rigorously defined semantics

- Simple - only one primitive: objects
- Expressive
  - can encode: classes and inheritance
  - but also functions (the lambda calculus)

Object Calculi

- Object-based (not class-based)
- Strongly-typed
- A Theory of Objects [Abadi & Cardelli ‘96]
- Object-based in practice: JavaScript, Self, etc.

Imperative Object Calculus

- Variant of the object calculus [Abadi & Leino, '04]
- Syntax

a, b	$::=$	x	variable
	$\mid$	let $x = a$ in b	variable binding
	$\mid$	$[f_i = x_i, m_j = \varsigma(y_j)b_j]_{i \in I, j \in J}$	object construction
	$\mid$	$x.f$	field selection
	$\mid$	$x.f := y$	field update
	$\mid$	$x.m$	method call
	$\mid$	true $\mid$ false	boolean constants
	$\mid$	if x then a else b	conditional
	$\mid$	$0 \mid$ succ $\mid \ldots$	numbers

- More syntactic sugar used in examples

Example Programs

- Factorial

$$[fac = \varsigma(y)\lambda n.\text{if } n = 0 \text{ then } 1 \text{ else } n \times y.fac(n-1)]$$

- Euclid's gcd algorithm

$$[gcd = \varsigma(y)\lambda x.\lambda z.\text{if } x < z \text{ then } y.gcd\ x\ (z-x) \text{ else if } z < x \text{ then } y.gcd\ (x-z)\ z \text{ else } x]$$

Operational Semantics

- Evaluate the program to get its meaning
- Abstract machine
- E.g. states of the machine = program + store
- Suffices for building a simple interpreter
- Simple, flexible and widely used
- Verification by testing, cannot be complete

Operational Semantics

- Small-step: evaluation = $\rightarrow^*$
- Reduction Relation: $\rightarrow \subseteq \mathbb{C}\text{onfig} \times \mathbb{C}\text{onfig}$
- Configurations: $\langle \sigma, a \rangle \in \mathbb{C}\text{onfig} = \mathbb{S}\text{tore} \times \mathbb{L}\text{Prog}$
- Reduction defined by rules

$$(\textsc{Red-Let1}) \quad \frac{\langle \sigma, a \rangle \rightarrow \langle \sigma', a' \rangle}{\langle \sigma, \mathsf{let}\ x\ =\ a\ \mathsf{in}\ b \rangle \rightarrow \langle \sigma', \mathsf{let}\ x\ =\ a'\ \mathsf{in}\ b \rangle}$$

$$(\textsc{Red-Let2}) \quad \langle \sigma, \mathsf{let}\ x\ =\ v\ \mathsf{in}\ b \rangle \rightarrow \langle \sigma, b[x \mapsto v] \rangle$$

$$(\textsc{Rel-Obj}) \quad \frac{l \notin dom\ \sigma \quad o = [f_i = v_i, m_j = b_j[y_j \mapsto l]]_{i \in I, j \in J}}{\langle \sigma, [f_i = v_i, m_j = \varsigma(y_j) b_j]_{i \in I, j \in J} \rangle \rightarrow \langle \sigma[l \mapsto o], l \rangle}$$

Example Reduction

$$\langle \emptyset, \mathsf{let}\ y\ =\ [m = \varsigma(x)x.m]\ \mathsf{in}\ y.m\rangle \rightarrow$$
$$\langle l = [m = l.m], \mathsf{let}\ y\ =\ l\ \mathsf{in}\ y.m\rangle \rightarrow$$
$$\langle l = [m = l.m], l.m\rangle \rightarrow$$
$$\langle l = [m = l.m], l.m\rangle \rightarrow \ldots$$

Nonterminating evaluation

Type Systems

- Static program analysis technique
- Conservative (sound but incomplete)
- In general:
  - Limited to safety properties
  - Limited form of reasoning about programs
  - Efficiently decidable (syntax driven)
- We will see an exception soon
- Types and Programming Languages [Pierce, '02]

Types of Objects

Conditional

$$\frac{E \vdash x : Bool \qquad E \vdash a_0 : A \qquad E \vdash a_1 : A}{E \vdash \mathit{if}\ x\ \mathit{then}\ a_0\ \mathit{else}\ a_1 : A}$$

Let

$$\frac{E \vdash a : A \qquad E, x\colon A \vdash b : B}{E \vdash \mathit{let}\ x = a\ \mathit{in}\ b : B}$$

Object construction $\qquad$ for $A \stackrel{syn}{=} [\mathrm{f}_i\colon A_i\ ^{i\in 1..n},\ \mathrm{m}_j\colon B_j\ ^{j\in 1..m}]$

$$\frac{E \vdash \diamond \qquad E \vdash x_i : A_i\ ^{i\in 1..n} \qquad E, y_j\colon A \vdash b_j : B_j\ ^{j\in 1..m}}{E \vdash [\mathrm{f}_i = x_i\ ^{i\in 1..n},\ \mathrm{m}_j = \varsigma(y_j)b_j\ ^{j\in 1..m}] : A}$$

Field selection

$$\frac{E \vdash x : [\mathrm{f}\colon A]}{E \vdash x.\mathrm{f} : A}$$

- Simple types: $A, B ::= Bool \mid Nat \mid [f : A, m : B]$
- Types of objects can be extended to specifications

Axiomatic Semantics

- Program meaning is what can be proved about it
- Focus on reasoning about program behavior
- Program logic
  - Deduction system for program correctness
  - E.g. Hoare logic [Floyd, '67] [Hoare, '69]

$$\frac{\vdash \{p \wedge b\} S_1 \{q\} \qquad \vdash \{p \wedge \neg b\} S_2 \{q\}}{\vdash \{p\} \text{if } b \text{ then } S_1 \text{ else } S_2 \{q\}}$$

- Widely used in program verification (E.g. Verisoft)

The Logic of Objects

- [Abadi & Leino, '04]
- A (undecidable) refinement of the type system
- Specifications generalize types

$$E \vdash b : A \rightsquigarrow E \vdash b : A :: \varphi$$

- φ is a first-order logic formula
- Seems hard to partially mechanize [Tang, '01]
- Soundness hard to prove
  [Schwinghammer & Reus, '06]

Higher-order Store

- Executable code can be stored
  - Pointers to functions in C
  - Callbacks in Java
  - General references in ML

- Recursion by "tying a knot in the store"

Higher-order Store

- Imperative object calculus
  - Dynamically allocated, higher-order store
  - Challenge: finding good semantic models
    - Reasoning about the behavior of programs
      - Program correctness (using a program logic)
- Operational Semantics
  - Useful for proving safety(progress&preservation)
  - Not suitable as the basis for a program logic
    [Benton, '05] [Benton, '06] [Schwinghammer, '06]

Denotational Semantics

- The meaning of a program is a mathematical object
- E.g. denotation of a lambda abstraction = function
- Higher-order store
  - Solving recursive domain equations
- Dynamic Allocation
  - Possible-world model =
    category of functors over cpos
- Domain theory, category theory, order theory
- More abstract - reasoning about program behavior

Denotational Semantics

- For the imperative object calculus
  - Complex [Reus & Schwinghammer, '06]
  - Separates logical validity from derivability
  - Specification = predicate on programs
  - Current models are still not abstract enough
    - Many natural equivalences do not hold

Step-indexed Semantics

- Foundational proof-carrying code by Appel et. all.
- Lambda calculus
  - Recursive types [Appel & McAllester, '01]
  - General references and polymorphism [Ahmed et. all., '03]
- Also for (very) low-level languages [Benton, '06]
- Based on a small-step operational semantics
- Type = set of indexed values
- "$e :_k T$ if e behaves like an element of T for k steps"

What We Are Working On

- Main goal: Develop a step-indexed semantics for the imperative object calculus with simple object types
  - Small-step semantics (done)
  - Safety for k steps
  - Types and store typings
    - Use indexing to solve cardinality paradoxes
  - Proving soundness of the typing rules

Possible Extensions

- Enriching the type system with:
  - Subtyping
  - Recursive object types [Schwinghammer, '06]
  - Impredicative polymorphism

Possible Extensions

- Defining a program logic extending types
  - Construct a sound deduction system
    - FOL assertions about the store
      [Abadi & Leino, '04] [Benton '05]
    - Dependent types+shallow embedding to HOL
      - e.g. Hoare Type Theory [Nanevski et all., '06]
- Local and modular reasoning
  - Separation Logic [Reynolds, '02]
  - Idealized ML [Krishnaswami et all., '06]

Advertising

More in-depth discussion
Master Honors Program Seminar
Monday 13th Nov. starting at 18:00

Thank you!

References

[1] Martín Abadi and Luca Cardelli. *A Theory of Objects*. Springer, 1996.

[2] Martín Abadi and K. Rustan M. Leino. A logic of object-oriented programs. In Nachum Dershowitz, editor, *Verification: Theory and Practice. Essays Dedicated to Zohar Manna on the Occasion of his 64th Birthday*, Lecture Notes in Computer Science, pages 11–41. Springer, 2004.

[3] Amal J. Ahmed, Andrew W. Appel, and Roberto Virga. An indexed model of impredicative polymorphism and mutable references. Princeton University, January 2003.

[4] Amal J. Ahmed, Matthew Fluet, and Greg Morrisett. A step-indexed model of substructural state. In Olivier Danvy and Benjamin C. Pierce, editors, *International Conference on Functional Programming (ICFP'05)*, pages 78–91. ACM Press, 2005.

[5] Andrew W. Appel and David McAllester. An indexed model of recursive types for foundational proof-carrying code. *ACM Transactions on Programming Languages and Systems*, 23(5):657–683, September 2001.

[6] Nick Benton. A typed, compositional logic for a stack-based abstract machine. In Zoltan Esik, editor, *Asian Symposium on Programming Languages and Systems APLAS'05*, volume 3780 of *Lecture Notes in Computer Science*, pages 182–196. Springer, 2005.

[7] Nick Benton. Abstracting allocation: the new new thing. In *Computer Science Logic*, volume 4207 of *Lecture Notes in Computer Science*, pages 364–380. Springer, 2006.

References

[8] Robert W. Floyd. Assigning meanings to programs. In Jacob T. Schwartz, editor, *Proceedings of Mathematical Aspects of Computer Science*, volume 19 of *Proceedings of Symposia in Applied Mathematics*, pages 19–32. American Mathematical Society, April 1967.

[9] C. A. R. Hoare. An Axiomatic Basis of Computer Programming. *Communications of the ACM*, 12:576–580, 1969.

[10] Neelakantan R. Krishnaswami, Lars Birkedal, Jonathan Aldrich, and John C. Reynolds. Idealized ML and its separation logic. Submitted, 2006.

[11] Aleksandar Nanevski, Amal Ahmed, Greg Morrisett, and Lars Birkedal. Abstract predicates and mutable adts in hoare type theory. Technical Report TR-14-06, Harvard University, 2006.

[12] Benjamin C. Pierce. *Types and Programming Languages*. The MIT Press, 2002.

[13] Bernhard Reus and Jan Schwinghammer. Denotational semantics for a program logic of objects. *Mathematical Structures in Computer Science*, 16(2):313–358, April 2006.

[14] John C. Reynolds. Separation logic: A logic for shared mutable data structures. In *Proceedings of the 17th IEEE Symposium on Logic in Computer Science*, pages 55–74. IEEE Computer Society, 2002.

[15] Jan Schwinghammer. *Reasoning about Denotations of Recursive Objects*. PhD thesis, Department of Informatics, University of Sussex, 2006.

[16] Francis Tang and Martin Hofmann. Generation of verification conditions for Abadi and Leino's logic of objects. Presented at 9th International Workshop on Foundations of Object-Oriented Languages, January 2002.