

## Semantics of Imperative Objects

**Context.** The object calculi of Abadi and Cardelli provide idealized models of object-oriented programming languages [1]. They have rigorously defined semantics, and they are simple since only objects are considered as primitives. At the same time they are expressive enough to encode all common features of practical (i.e., class-based) object-oriented programming languages like classes, subtyping and inheritance. In this work we will study the semantics of a variant of Abadi and Cardelli's imperative object calculus, as presented by Abadi and Leino [2]. This calculus is particularly interesting since it combines objects with *dynamically allocated, higher-order store*.

While higher-order store is present in different forms in almost all practical programming languages (pointers to functions in C, callbacks in Java, or general references in ML), it is challenging to find good semantic models in which one can reason about the behaviour of programs. *Syntactic arguments*, based solely on the operational semantics, suffice to prove properties such as type preservation, but are not suitable as a basis for program logics like that of Abadi and Leino [2]. We believe that specifications of program behaviour should have a meaning *independent* of the particular proof system on which syntactic preservation proofs rely [7, 6, 10]. On the other hand, a "classical" denotational semantics of higher-order store based on partial orders tends to become rather complex. In fact, modelling dynamic allocation alone usually means that one has to move to a *possible-world model*, formalized as a category of functors over *cpos*. While this achieves the goal of separating the notion of logical validity from derivability, the known models are not very abstract in that many natural equivalences involving state do not hold.

An alternative is to use a *step-indexed semantics*, an approach developed by Appel and his collaborators in the context of foundational proof-carrying code [5]. Based on a small-step operational semantics, types are interpreted as sets of indexed values. Informally, an expression has a certain type if it behaves like an element of that type for a fixed number of steps. The usual type inference rules then become derived lemmas, and type safety of the operational semantics is an immediate consequence of this interpretation of types.

A step-indexed semantics has been introduced for lambda calculus with recursive and polymorphic types in [5]. Later this has been successfully extended to an imperative language with general references and impredicative polymorphism [3], substructural state [4], and has also been used for low-level languages [5, 7].

**Task description.** The aim of this thesis is to develop a step-indexed semantics for the imperative object calculus. The main goal is to define such a semantics for simple object types, and prove the usual type inference rules sound. Once this has been achieved, there are several extensions of the construction that can be investigated:

- Enriching the type system with features such as *subtyping*, *recursive object types* or *impredicative polymorphism*.
- Defining a *program logic* for the imperative object calculus, along the lines of [2]. The logic may then even be adapted to *local* and *modular reasoning* in the style of separation logic [9, 8].

**Time frame.** November 2006 - April 2007

**Supervisor.** Jan Schwinghammer (jan@ps.uni-sb.de)

**Responsible Professor.** Prof. Dr. Gert Smolka

## 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.
- [8] Neelakantan R. Krishnaswami, Lars Birkedal, Jonathan Aldrich, and John C. Reynolds. Idealized ML and its separation logic. Submitted, 2006.
- [9] 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.
- [10] Jan Schwinghammer. *Reasoning about Denotations of Recursive Objects*. PhD thesis, Department of Informatics, University of Sussex, 2006.