History of Object Calculi

Ahmed Abdelmeged

March 29, 2010

1 Introduction

The main purpose of object calculi is to formally model Object Oriented Programming Languages as faithfully as possible. Object calculi typically take the form of some typed extension of the lambda calculus. For some calculi types are syntactic objects. For others types get their meaning from semantic domains.

2 Phase I: Record Calculi

Records are used to model objects. A record is a function from labels to values. With record calculi, the set of messages that can be sent to objects is bounded by the set of its labels. Furthermore, records do not faithfully model objects.

Two milestones in this phase are [3] and [8]. In [3], objects are modelled as read-only, recursive, fixed size records. Types are syntactic objects. A syntactic subtyping relation is defined on type expressions.

In [8], objects are modelled as extensible records. Types are syntactic objects. Parametric subtyping (let polymorphism) is used rather than inclusion polymorphism.

3 Phase II: Faithful Encoding of Objects

Structural subtyping does only takes into account the names and types of methods (and fields) of an object. A more faithful model of inheritance should take into account how a method ended up in an object. Furthermore, field update (method override) is an important feature of OOPLs that needs to be supported by the calculus.

3.1 Recursive Record Semantics

Recursive record semantics are suitable for modelling class based languages as the object’s self is frozen. Cook et al. [4] presented a model for inheritance based on record generators. However, once records are generated, they are not updatable. [6] presented a model that distinguishes fields from methods. Fields are updatable after record generation.
3.2 Self Application Semantics

Self application can be used for modelling both class-based as well as prototype-based OOPLs [5]. Typed encoding of object calculi adopting the self application semantics proved challenging. In [1], Abadi and Cardelli, presented a typed object calculus. However, types were not syntactic.

4 Phase III: A Theory of Primitive Objects

In [2], Abadi and Cardelli presented a calculus for primitive objects that combines object subsumption with method override. Furthermore, types are syntactic. In the same paper, they shown how to encode the lambda calculus into their calculus. They did not support extensible objects.

In [7] Remey, extended Abadi and Cardelli’s calculus to support extensible objects.

References


