Objects and subtyping in the λΠ-calculus modulo

Ali Assaf[1,2], Raphaël Cauderlier[1,3], and Catherine Dubois[3,4]

[1] INRIA Paris-Rocquencourt, Paris, France
[2] École Polytechnique, Paris, France
[3] CNAM, Paris, France
[4] ENSIIE, Évry, France

In this talk, we present a shallow embedding of an object calculus, the ς-calculus, in the λΠ-calculus modulo. The main difficulty is the encoding of subtyping. We propose a solution that makes use of rewriting in order to ease the handling of subtyping proofs.

Motivations The λΠ-calculus modulo is an extension of the λΠ-calculus [7] with rewrite rules. Implemented in the Dedukti type-checker [9], it can be used as a logical framework for the implementation of formal systems [5]. In this framework, rewrite rules can be introduced in addition to β-reduction to extend the conversion relation between terms.

Cousineau and Dowek [4] showed that any functional pure type system can be encoded in the λΠ-calculus modulo using appropriate rewrite rules. The main emphasis of this embedding is that it is *shallow*, as opposed to *deep* embeddings. As much as possible, the features of the object language are implemented by the corresponding features in the meta-language: bindings are represented using binders, typing using typing, reduction using reduction, etc. Besides avoiding the reimplementation of these features, shallow embeddings have the advantage of being more compact, more readable, and more efficient than deep embeddings.

While encoding languages with functional features in the λΠ-calculus modulo seems natural, encoding object-oriented languages, that share no feature with the λΠ-calculus modulo, is less obvious. In particular, encoding subtyping is a challenging problem, because it is absent from the target language. In the λΠ-calculus modulo, each term has a unique type. If M has type A and A is not convertible to B then M does not have type B. Moreover, it is not possible to rewrite A to B, as then any term of type B would also have type A, which would be unsound.

Related work A lot of work has been done in the field of encoding of objects. Several such encodings in System $\mathrm{F}^{\omega}_{<:}$ have been proposed and compared [8, 2]. They often rely on existential types and some form of recursion.

In 1996, Abadi and Cardelli [1] defined several object calculi which consider objects as a primitive notion instead of encoding them in a λ-calculus. These calculi are very primitive in the sense that they can be used to represent both object-based and class-based languages and they do not distinguish methods from fields. These calculi have been used as examples for testing the effectivity of deep embeddings in the Coq proof assistant [6, 3].

One of these calculi is the simply-typed ς-calculus. It represents objects as records of the form $[l_i = \varsigma(x : A)t_i]_{i=1\ldots n}$, and each method has only one parameter, introduced by the ς binder, which represents *self*. This calculus has simple typing rules and operational semantics.

$$\frac{\Gamma, x : A \vdash t_i : A_i \quad \forall i = 1 \ldots n}{\Gamma \vdash t : A} \quad \text{where } A = [l_i : A_i] \text{ and } t = [l_i = \varsigma(x : A)t_i]_{i=1\ldots n}$$

$$\begin{array}{lcll} t.l_j & \longrightarrow & t_j\{x := t\} & \text{(method selection)} \\ t.l_j \Leftarrow \varsigma(x : A)u & \longrightarrow & t\ \{l_j := \varsigma(x : A)u\} & \text{(method update)} \end{array}$$

Subtyping is defined by $[l_i : A_i]_{i=1\ldots n+m} <: [l_i : A_i]_{i=1\ldots n}$, so A is a subtype of B if and only if A and B coincide on the labels of B. With its minimalist definition, the simply-typed ς-calculus is an ideal candidate for the study of encodings of object-oriented mechanisms.

Contributions We give an encoding of the simply-typed ς-calculus in the $\lambda\Pi$ calculus modulo. We encode types and objects using association lists. Since sub-lists of well-typed objects need not be well-typed, we have to introduce partially constructed (ill-typed) objects. Selection and update of methods are performed using

$$\begin{array}{lcl} \mathsf{select} & : & \Pi A : \mathsf{type}, \mathsf{Object}\ A \to \Pi l : \mathsf{Label}, \mathsf{Object}\ (\mathsf{assoc}\ A\ l) \\ \mathsf{update} & : & \Pi A : \mathsf{type}, \mathsf{Object}\ A \to \Pi l : \mathsf{Label}, (\mathsf{Object}\ A \to \mathsf{Object}\ (\mathsf{assoc}\ A\ l)) \to \mathsf{Object}\ A \end{array}$$

and the operational semantics is translated to the following rewrite rules:

$$\begin{array}{lrcl} \mathsf{select} & A\ [l = m, \ldots]\ l & \hookrightarrow & m\ [l = m, \ldots] \\ \mathsf{update} & A\ [l = m, \ldots]\ l\ m' & \hookrightarrow & [l = m', \ldots] \end{array}$$

We use explicit coercions to handle subtyping. The coercion function

$$\mathsf{coerce} : \Pi A, B : \mathsf{type}, \mathsf{proof}(A <: B) \to \mathsf{Object}\ A \to \mathsf{Object}\ B$$

takes an extra logical argument of type $\mathsf{proof}(A <: B)$. We show how to make special use of rewrite rules and reflection to reduce $\mathsf{proof}(A <: B)$ to $\mathsf{proof}\ \top$ and thus avoid carrying big subtyping proofs.

This encoding has been implemented in Dedukti and tested on the examples from Abadi and Cardelli [1] which illustrate all the features of the simply-typed ς-calculus. Our implementation can be found online at: `https://www.rocq.inria.fr/deducteam/Sigmaid`.

References

[1] M. Abadi and L. Cardelli. *A Theory of Objects*. Monographs in Computer Science. Springer New York, 1996.

[2] K. B. Bruce, L. Cardelli, and B. C. Pierce. Comparing object encodings. *Information and Computation*, 155(1/2):108–133, November 1999.

[3] A. Ciaffaglione, L. Liquori, and M. Miculan. Reasoning about object-based calculi in (co)inductive type theory and the theory of contexts. *J. Autom. Reasoning*, 39(1):1–47, 2007.

[4] D. Cousineau and G. Dowek. Embedding pure type systems in the lambda-pi-calculus modulo. In S. Ronchi Della Rocca, editor, *TLCA*, volume 4583 of *LNCS*, pages 102–117. Springer, 2007.

[5] G. Dowek. A theory independent curry-de bruijn-howard correspondence. In *Proceedings of the 39th International Colloquium Conference on Automata, Languages, and Programming - Volume Part II*, ICALP'12, pages 13–15, Berlin, Heidelberg, 2012. Springer-Verlag.

[6] G. Gillard. A formalization of a concurrent object calculus up to alpha-conversion. In D. A. McAllester, editor, *CADE*, volume 1831 of *LNCS*, pages 417–432. Springer, 2000.

[7] R. Harper, F. Honsell, and G. Plotkin. A framework for defining logics. *J. ACM*, 40(1):143–184, January 1993.

[8] B. C. Pierce and D. N. Turner. Simple type-theoretic foundations for object-oriented programming. *Journal of Functional Programming*, 4(2):207–247, April 1994.

[9] R. Saillard. Dedukti: a universal proof checker. In *Foundation of Mathematics for Computer-Aided Formalization Workshop*, Padova, 2013.