# Monadic Presentations of Lambda Terms Using Generalized Inductive Types

Thorsten Altenkirch and Bernhard Reus

Ludwig-Maximilians-Universität, Oettingenstr. 67, 80538 München, Germany,
{alti,reus}@informatik.uni-muenchen.de
Phone: +49 89 2178 {2209,2178} Fax: +49 89 2178 {2238,2175}

**Abstract.** We present a definition of untyped $\lambda$-terms using a heterogeneous datatype, i.e. an inductively defined operator. This operator can be extended to a Kleisli triple, which is a concise way to verify the *substitution laws* for $\lambda$-calculus. We also observe that repetitions in the definition of the monad as well as in the proofs can be avoided by using well-founded recursion and induction instead of structural induction. We extend the construction to the simply typed $\lambda$-calculus using dependent types, and show that this is an instance of a generalization of Kleisli triples. The proofs for the untyped case have been checked using the LEGO system.
**Keywords.** Type Theory, inductive types, $\lambda$-calculus, category theory.

## 1 Introduction

The metatheory of substitution for $\lambda$-calculi is interesting maybe because it seems intuitively obvious but becomes quite intricate if we take a closer look. [Hue92] states seven formal properties of substitution which are then used to prove a general substitution theorem. When formalizing the proof of strong normalisation for System F [Alt93b,Alt93a] the first author formally verified five substitution properties quite similar to those of [Hue92].

Therefore it seems a good idea to look for a more general and elegant way to state and verify the substitution laws. Obviously, this is also related to the way lambda terms are presented.

We find a partial answer in the work of Bellegarde and Hook [BH94] who take the view that lambda terms should be represented by an operator $\mathrm{Lam} \in \mathbf{Set} \to \mathbf{Set}$, where **Set** denotes the universe of sets, such that $\mathrm{Lam}(X)$ is the set of $\lambda$-terms with variables in $X$. This corresponds to the presentation of terms in universal algebra as an operator $\mathrm{Term} \in \mathbf{Set} \to \mathbf{Set}$. The substitution laws are captured by verifying that Lam can be extended to a monad or equivalently to a *Kleisli triple* (cf. Section 2.1, see also [Man76,Mog91]).

In this paper we are going to revise and extend the work of Bellegarde and Hook in the following ways:

J. Flum and M. Rodríguez-Artalejo (Eds.): CSL'99, LNCS 1683, pp. 453–468, 1999.

- The presentation of Lam, see Section 3.2, is improved by using a *heterogeneous datatype*[1], i.e. there are no meaningless terms in our representation. Heterogeneous datatypes have already been discussed in [BM98], where they are called *nested datatypes* and modelled by initial algebras in functor categories, which seems unsatisfactory. Building on this approach, in [BP99] heterogeneous definitions of untyped $\lambda$-terms are investigated.
- Repetitions in the definition of the monad and in the verification can be avoided by using well founded recursion (along a primitive recursive well-ordering) instead of structural recursion, see section 4.
- The development has been verified using the LEGO system, see section 4.5.
- We also extend this approach to the simply typed $\lambda$-calculus, see Section 5. To do this we present a generalization of Kleisli triples, which we call Kleisli structures, see 5.1.
- We analyze the type of inductive definitions needed in every step of the formalization using initial algebras of functors. We consider two generalizations of the usual scheme of inductive definitions: heterogeneous (see 3.1) and dependent inductive definitions (see Section 5.2).

Our work seems to be closely related to recent work by Fiore, Plotkin and Turi [FPT99] who pursue a more abstract algebraic treatment of signatures with binders but do not cover the simply typed case. Higher order syntax can also be used to represent $\lambda$-terms, i.e. in [Hof99].

## 2 Preliminaries

As a metatheory we use an informal version of extensional Type Theory, details can be found in [Mar84,Hof97]. Since we do not exploit the proposition-as-types principle we work in a system quite close to conventional intuitionistic set theory. We use **Set** and **Prop** to denote the types of sets and propositions.

Notationally, we adopt the following conventions: We write the type of implicit parameters of dependent functions as subscripts, i.e. $\Pi_{n\in\mathrm{Nat}}\mathrm{Fin}(n) \to \mathbf{Set}$ is a type of functions whose first argument is usually omitted. The hidden argument can be made explicit by putting it in subscript, i.e. we write e.g. $f_X \in T(X)$ when we mean $f \in \Pi_{X\in C}T(X)$ for some type $C \in \mathbf{Set}$ obvious from the context. Given $P, Q \in A \to \mathbf{Prop}$ we write $P \subseteq Q$ for $\forall a \in A.P(a) \to Q(a)$. Given a curried function $f \in A_1 \to A_2 \to \ldots \to A_n$ we write the application to a sequence of arguments $a_1, a_2, \ldots, a_n$ as $f(a_1, a_2, \ldots, a_n)$. The same convention holds for $\Pi$-types.

The rest of this section briefly reviews Kleisli triples, initial algebras, and inductive datatypes and might be skipped by the experienced reader.

### 2.1 Kleisli Triples

We present monads as Kleisli triples, i.e.

[1] It seems that the idea for this presentation goes back to Hook, but he didn't use it in the paper because it cannot be implemented in SML.