Finite axiomatizations of inductive and inductive-recursive definitions

Peter Dybjer and Anton Setzer

May 14, 1998

Abstract

We first present a finite axiomatization of strictly positive inductive types in the simply typed lambda calculus. Then we show how this axiomatization can be modified to encompass simultaneous inductive-recursive definitions in intuitionistic type theory. A version of this has been implemented in the *Half* system which is based on Martin-Lf's logical framework.

1 Introduction

The present note summarizes a presentation to be given at the Workshop on Generic Programming, Marstrand, Sweden, June 18th, 1998.

We use Martin-Löf's logical framework as a metalanguage for axiomatizing inductive definitions in the simply typed lambda calculus. We also show how to generalize this axiomatization to the case of inductive-recursive definitions in the lambda calculus with dependent types. The reader is referred to the full paper [7] for a more complete account focussing on induction-recursion. Related papers discussing inductive definitions in intuitionistic type theory include Backhouse [1, 2], Coquand and Paulin [3], Dybjer [5, 4], and Paulin [**?**] For an introduction to inductive-recursive definitions see Dybjer [6].

2 Inductive definitions in the simply typed lambda calculus

2.1 Inductive types as initial algebras

Let us first consider the question of how to formalize inductive types in the setting of the simply typed λ-calculus. We shall consider *generalized inductive definitions* of types given by a finite number of constructors

$$\mathrm{intro} : \Phi \mathrm{U} \to \mathrm{U},$$

where Φ is *strictly positive* in the following restricted sense

- The constant functor $\Phi D = 1$ is strictly positive. This is the base case corresponding to an introduction rule with no premises.

- If Ψ is strictly positive and A is a (small) type, then $\Phi D = A \times \Psi D$ is strictly positive. This corresponds to the addition of a *non-recursive* premise.
- If Ψ is strictly positive and B is a (small) type, then $\Phi D = (B \to D) \times \Psi D$ is strictly positive. This corresponds to the addition of a *recursive* premise, where B corresponds to the hypotheses of this premise in a generalized inductive definition.

Note that all occurrences of U in ΦU are *strictly positive* in the standard sense that U does not occur to the left of an arrow in ΦU.

Recall that such inductive types can be captured categorically as initial Φ-algebras for an endofunctor Φ on a category of types. An initial Φ-algebra is an arrow

$$\Phi \mathrm{U} \xrightarrow{\text{intro}} \mathrm{U}$$

such that for any other Φ-algebra

$$\Phi D \xrightarrow{d} D$$

there is a unique arrow $\mathrm{T} : \mathrm{U} \longrightarrow C$, such that the following diagram commutes

$$\begin{tikzcd}
\Phi \mathrm{U} \arrow[r, "\text{intro}"] \arrow[d, "\Phi \mathrm{T}"'] & \mathrm{U} \arrow[d, "\mathrm{T}"] \\
\Phi D \arrow[r, "d"] & D
\end{tikzcd}$$

U is the type inductively generated by Φ and T is a function defined by *iteration*. In order to obtain definition by primitive or structural recursion we consider the following diagram:

$$\begin{tikzcd}
\Phi \mathrm{U} \arrow[r, "\text{intro}"] \arrow[d, "x \mapsto (x, \Phi \mathrm{R}\, x)"'] & \mathrm{U} \arrow[d, "y \mapsto (y, \mathrm{R}\, y)"] \\
\Phi \mathrm{U} \times \Phi D \arrow[r, "(x,z) \mapsto (\text{intro}\, x, d\, x\, z)"] & \mathrm{U} \times D
\end{tikzcd}$$

and the equation

$$\mathrm{R}\,(\text{intro}\, a) = d\, a\, (\Phi \mathrm{R}\, a)$$

2.2 Type-theoretic formalization

Call inductive types "sets" and call a type generated from sets by $\to$, $\times$, and 1, a "small type".

To obtain a formal system for inductive types, we first inductively generate the type SP of codes for strictly positive functors. To each code ϕ we associate an object part Arg_ϕ and the arrow part map_ϕ of the strictly positive functor in question. Moreover, a function $d : \mathrm{Arg}_\phi\, D \to D$ for some $\phi : \mathrm{SP}$ is called a (possibly infinitary) D-operator.

With this new notation the initial algebra diagram becomes:

$$\begin{tikzcd}
\mathrm{Arg}_\phi \mathrm{U} \arrow[r, "\mathrm{intro}"] \arrow[d, "\mathrm{map}_{\phi,\mathrm{U},D}\,\mathrm{T}"'] & \mathrm{U} \arrow[d, "\mathrm{T}"] \\
\mathrm{Arg}_\phi D \arrow[r, "d"] & D
\end{tikzcd}$$

We thus have the following three formation rules:

$$\mathrm{SP}\ \text{type}$$

$$\frac{\phi : \mathrm{SP} \qquad D\ \text{type}}{\mathrm{Arg}_\phi\, D\ \text{type}}$$

$$\frac{\phi : \mathrm{SP} \qquad U, D\,\text{type} \qquad T : U \to D}{\mathrm{map}_{\phi,U,D}\, T : \mathrm{Arg}_\phi\, U \to \mathrm{Arg}_\phi\, D}$$

(We also have rules which state that Arg, map, etc preserve equality, but we will not spell them out here.)

We have the following introduction rules for SP:

$$\mathrm{nil} : \mathrm{SP}$$

$$\frac{A : \mathrm{stype} \qquad \phi : \mathrm{SP}}{\mathrm{nonrec}\, A\, \phi : \mathrm{SP}}$$

$$\frac{B : \mathrm{stype} \qquad \phi : \mathrm{SP}}{\mathrm{rec}\, B\, \phi : \mathrm{SP}}$$

and the corresponding equality rules for Arg

$$\begin{aligned}
\mathrm{Arg}_{\mathrm{nil}}\, D &= 1 \\
\mathrm{Arg}_{\mathrm{nonrec}\, A\, \phi}\, D &= A \times (\mathrm{Arg}_\phi\, D) \\
\mathrm{Arg}_{\mathrm{rec}\, B\, \phi}\, D &= (B \to D) \times (\mathrm{Arg}_\phi\, D)
\end{aligned}$$

and map

$$\begin{aligned}
\mathrm{map}_{\mathrm{nil}\, C\, D}\, T\, * &= * \\
\mathrm{map}_{(\mathrm{nonrec}\, A\, \phi), C, D}\, T\, (\gamma, a) &= (\mathrm{map}_{\phi\, C\, D}\, T\, \gamma, a) \\
\mathrm{map}_{(\mathrm{rec}\, B\, \phi), C, D}\, T\, (\gamma, f) &= (\mathrm{map}_{\phi, C, D}\, T\, \gamma, T \circ f)
\end{aligned}$$

An inductive type (a "set") is now given by a finite list of codes for strictly positive operators, that is, by an object of SP^n for some number n. We can now give the formation, introduction, elimination, and equality rules for the set U_ρ given by $\rho : \mathrm{SP}^n$.

We thus have the following formation rule for inductive types

$$\frac{\rho : \mathrm{SP}^n}{\mathrm{U}_\rho : \mathrm{set}}$$

Moreover, we have the introduction rule

$$\frac{\rho : \mathrm{SP}^n \qquad\qquad i : N_n}{\mathrm{intro}_{\rho,i} : \mathrm{Arg}_{\rho\, i}\, \mathrm{U}_\rho \to \mathrm{U}_\rho}$$

Let $\rho : \mathrm{SP}^n$ and D type be global premises in the elimination and equality rules.

The elimination rule for iteration is

$$\frac{d : (i : N_n) \to (\mathrm{Arg}_{\rho\, i}\, D) \to D}{\mathrm{T}_\rho\, d : \mathrm{U}_\rho \to D}$$

The equality rule for iteration is

$$\frac{d : (i : N_n) \to (\mathrm{Arg}_{\rho\, i}\, D) \to D \qquad\qquad i : N_n \qquad\qquad a : \mathrm{Arg}_{\rho\, i}\, \mathrm{U}_\rho}{\mathrm{T}_\rho\, d\, (\mathrm{intro}_{\rho,i}\, a) = d\, i\, (\mathrm{map}_{(\rho\, i)\, \mathrm{U}\, D}\, (\mathrm{T}_\rho\, d)\, a) : D}$$

The elimination rule for primitive recursion is

$$\frac{d : (i : N_n) \to (\mathrm{Arg}_{\rho\, i}\, \mathrm{U}) \to (\mathrm{Arg}_{\rho\, i}\, D) \to D}{\mathrm{R}_\rho\, d : \mathrm{U}_\rho \to D}$$

The equality rule for primitive recursion is

$$\frac{d : (i : N_n) \to (\mathrm{Arg}_{\rho\, i}\, \mathrm{U}) \to (\mathrm{Arg}_{\rho\, i}\, D) \to D \qquad\qquad i : N_n \qquad\qquad a : \mathrm{Arg}_{\rho\, i}\, \mathrm{U}_\rho}{\mathrm{R}_\rho\, d\, (\mathrm{intro}_{\rho,i}\, a) = d\, i\, a\, (\mathrm{map}_{(\rho\, i)\, \mathrm{U}\, D}\, (\mathrm{R}_\rho\, d)\, a) : D}$$

3 Inductive-recursive definitions

3.1 Informal discussion

In the case of a simultaneous inductive-recursive definition the domain of the constructor intro may depend on $\mathrm{T} : \mathrm{U} \to D$ as well as on U Therefore, it also (indirectly) depends on D. If we write out these new dependencies in the diagram for initial algebras we get the following tentative picture

$$\begin{tikzcd}
\Phi DIR \arrow[r, "\mathrm{intro}"] \arrow[d, "\Phi DIR"'] & I \arrow[d, "R"] \\
\Phi D \arrow[r, "d"] & D
\end{tikzcd}$$

Think of D as a type of "semantic" objects and of $d : \Phi D \to D$ as a (possibly infinitary) "semantic" operation with ΦD as the generalized "arity" of d. U is a universe of codes for objects in D and $\mathrm{T} : \mathrm{T} \to D$ is the decoding function. The (possibly infinitary) constructor intro is the syntactic reflection of $d : \Phi D \to D$.

Note that Φ is no longer an endofunctor on the category of types, but rather consists of a triple, since what corresponds to the object part now is split up into

- a "semantic" part $D \mapsto \Phi D$ which returns the source (generalized arity) of d;

- and a "syntactic" part $D, I, R \mapsto \Phi DIR$ which returns the source of intro.

Moreover, the modified class (of codes for such triples) $\mathrm{SP} = \mathrm{SP}\, D$ depends on the semantic type D and needs to be defined simultaneously with the decoding function $-$ which returns the codomain of a semantic operation. SP and $-$ together specify what it means to be a (possibly infinitary) operation on D in the framework of dependent types.

3.2 Type-theoretic formalization

We shall now give the formal rules for inductive-recursive definitions. Such a definition is always parameterized with respect to a type D: a particular inductive-recursive definition defines a universe for a finite number of D-operators.

So first we need to define what a D-operator is. To this end we simultaneously introduce the type $\mathrm{SP}\, D$ of codes for arities (codomains) of D-operators and the associated decoding function

$$\frac{D\,\mathrm{type} \qquad \phi : \mathrm{SP}\, D}{\mathrm{Arg}\ \phi\,\mathrm{type}}$$

which returns the codomain (coded by some $\phi : \mathrm{SP}\, D$) of a D-operator. Note that, since the type of ϕ depends on D, we can no longer have D as an argument to Arg which comes after ϕ, as was the case for the simply typed case where ϕ indeed coded a functor.

$$\begin{tikzcd}
\mathrm{arg}_{D,\delta}\,\mathrm{U}\,\mathrm{T} \arrow[r, "\mathrm{intro}"] \arrow[d, "\mathrm{map}_{D,\delta}\,\mathrm{U}\,\mathrm{T}"'] & \mathrm{U} \arrow[d, "\mathrm{T}"] \\
\mathrm{Arg}_{D,\delta} \arrow[r, "d"] & D
\end{tikzcd}$$

$$\frac{D \text{ type}}{\mathrm{SP}_D \text{ type}}$$

$$\frac{D \text{ type} \qquad \delta : \mathrm{SP}_D}{\mathrm{Arg}_{D,\delta} \text{ type}}$$

$$\frac{D \text{ type} \qquad \delta : \mathrm{SP}_D \qquad U \text{ set} \qquad T : U \to D}{\mathrm{arg}_{D,\delta}\, U\, T \text{ stype}}$$

$$\frac{D \text{ type} \qquad \delta : \mathrm{SP}_D \qquad U \text{ set} \qquad T : U \to D}{\mathrm{map}_{D,\delta}\, U\, T : (\mathrm{arg}_{D,\delta}\, U\, T) \to \mathrm{Arg}_{D,\delta}}$$

From now on D type is a global premise.

$$\mathrm{nil} : \mathrm{SP}_D$$

$$\frac{A \text{ stype} \qquad \delta : A \to \mathrm{SP}_D}{\mathrm{nonrec}\, A\, \delta : \mathrm{SP}_D}$$

$$\frac{B \text{ stype} \qquad \delta : (B \to D) \to \mathrm{SP}_D}{\mathrm{rec}\, B\, \delta : \mathrm{SP}_D}$$

$$
\begin{array}{rcl}
\mathrm{Arg}_{D,\mathrm{nil}} & = & 1 \\
\mathrm{Arg}_{D,\mathrm{nonrec}\,A\,\delta} & = & (x : A) \times \mathrm{Arg}_{D,\delta\,x} \\
\mathrm{Arg}_{D,\mathrm{rec}\,B\,\delta} & = & (f : B \to D) \times \mathrm{Arg}_{D,\delta\,f}
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{arg}_{D,\mathrm{nil}}\,U\,T & = & 1 \\
\mathrm{arg}_{D,\mathrm{nonrec}\,A\,\delta}\,U\,T & = & (x : A) \times (\mathrm{arg}_{D,\delta\,x}\,U\,T) \\
\mathrm{arg}_{D,\mathrm{rec}\,B\,\delta}\,U\,T & = & (f : B \to U) \times (\mathrm{arg}_{D,\delta\,(T\circ f)}\,U\,T)
\end{array}
$$

$$
\begin{array}{rcl}
\mathrm{map}_{D,\mathrm{nil}}\,U\,T\,* & = & * \\
\mathrm{map}_{D,\mathrm{nonrec}\,A\,\delta}\,U\,T\,(a,\gamma) & = & (a, \mathrm{map}_{D,\delta\,a}\,U\,T\,\gamma) \\
\mathrm{map}_{D,\mathrm{rec}\,B\,\delta}\,U\,T\,(f,\gamma) & = & (T \circ f, \mathrm{map}_{D,\delta\,(T\circ f)}\,U\,T\,\gamma)
\end{array}
$$

References

[1] R. Backhouse. On the meaning and construction of the rules in Martin-Löf's theory of types. In *Proceedings of the Workshop on General Logic, Edinburgh, February 1987.* Laboratory for Foundations of Computer Science, Department of Computer Science, University of Edinburgh, 1988. ECS-LFCS-88-52.

[2] R. Backhouse, P. Chisholm, G. Malcolm, and E. Saaman. Do-it-yourself type theory (part 1). *Formal Aspects of Computing*, pages 19–84, 1989.

[3] T. Coquand and C. Paulin. Inductively defined types, preliminary version. In *LNCS 417, COLOG '88, International Conference on Computer Logic.* Springer-Verlag, 1990.

[4] P. Dybjer. Inductive sets and families in Martin-Löf's type theory and their set-theoretic semantics. In *Logical Frameworks*, pages 280–306. Cambridge University Press, 1991.

[5] P. Dybjer. Inductive families. *Formal Aspects of Computing*, pages 440–465, 1994.

[6] P. Dybjer. A general formulation of simultaneous inductive-recursive definitions in type theory. *Journal of Symbolic Logic*, 1998. To appear.

[7] P. Dybjer and A. Setzer. A finite axiomatization of inductive-recursive definitions. Draft paper, April 1998.

[8] C. Paulin-Mohring. Inductive definitions in the system Coq - rules and properties. In *Proceedings Typed λ-Calculus and Applications*, pages 328–245. Springer-Verlag, LNCS, March 1993.