Dialgebra-inspired Syntax for Dependent Inductive and Coinductive Types

Henning Basold[1,2] and Herman Geuvers[1,3]

[1] Radboud University, iCIS, Intelligent Systems
[2] CWI, Amsterdam, The Netherlands
[3] Technical University Eindhoven, The Netherlands

The goal of this work is the development of a syntax that treats inductive and coinductive types with term dependencies on a par. We achieve this by extending Hagino's idea [Hag87] to dependent types, in that we interpret data types as dialgebras in fibrations. This follows the ideas we present in the talk "Dependent Inductive and Coinductive Types Through Dialgebras in Fibrations". We note that our syntax for inductive types is similar to GADTs [HF11], and that coinductive types are defined by their destructors like in [APTS13].

The context, types and terms of our calculus are introduced using the following judgements.

- $\vdash \Delta \mid \Gamma\ \mathbf{ctx}$ – The type variable context Δ and term variable context Γ are well-formed.
- $\Delta \mid \Gamma \vdash A : *$ – The type A is well-formed in the context $\Delta \mid \Gamma$.
- $\Gamma \vdash t : A$ – The term t is well-formed and of type A in the context Γ.
- $f : \Gamma_1 \to \Gamma_2$ – The context morphism f is well-formed with type $\Gamma_1 \to \Gamma_2$.

The context judgement is given by the following rules.

$$\frac{}{\vdash \emptyset \mid \emptyset\ \mathbf{ctx}} \qquad \frac{\emptyset \mid \Gamma \vdash A : *}{\vdash \Delta \mid \Gamma, x : A\ \mathbf{ctx}} \qquad \frac{\vdash \Delta \mid \Gamma'\ \mathbf{ctx} \qquad \vdash \Delta \mid \Gamma\ \mathbf{ctx}}{\vdash \Delta, (\Gamma' \vdash X : *) \mid \Gamma\ \mathbf{ctx}}$$

It is important to note that, whenever a term variable is introduced into a context, its type is not allowed to use free type variables. This ensures that types are strictly positive.

The notion of substitution, we use, builds on *context morphisms* that are formed as follows.

$$\frac{\vdash \Delta \mid \Gamma\ \mathbf{ctx}}{() : \Gamma \to \emptyset} \qquad \frac{f : \Gamma_1 \to \Gamma_2 \qquad \Gamma_1 \vdash t : A[f]}{(f, t) : \Gamma_1 \to (\Gamma_2, x : A)}$$

Explicitly, if $\Gamma_2 = x_1 : A_1, \ldots, x_n : A_n$, then a context morphism $f : \Gamma_1 \to \Gamma_2$ is a tuple $f = (f_1, \ldots, f_n)$ with $\Gamma_1 \vdash f_i : A_i[(f_1, \ldots, f_{i-1})]$ for each $i = 1, \ldots, n$.

The judgement for type construction is given, together with the usual contraction, weakening and exchange rules for type variables, by the following rules.

$$\frac{}{\Delta, (\Gamma' \vdash X : *) \mid \Gamma, \Gamma' \vdash X : U_i}\ \mathbf{(TyVar\text{-}I)} \qquad \frac{\Delta \mid \Gamma_1 \vdash A : * \qquad g : \Gamma_2 \to \Gamma_1}{\Delta \mid \Gamma_2 \vdash A[g] : *}\ \mathbf{(Subst\text{-}Ty)}$$

$$\frac{\Delta, (\Gamma \vdash X : *) \mid \Gamma_k \vdash A_k : * \qquad f_k : \Gamma_k \to \Gamma \qquad k = 1, \ldots, n \qquad \rho \in \{\mu, \nu\}}{\Delta \mid \Gamma \vdash \rho(X; \vec{f}; \vec{A}) : *}\ \mathbf{(FP\text{-}Ty)}$$

Note that substitutions with context morphisms are part of the syntax and that type variables come with a term variable context. These contexts determine the context in which an initial/-final dialgebra lives. The dialgebras essentially bundle the *local* context, the domain and the codomain of their constructors respectively destructors, as we will see.

This brings us to the rules for term constructions, the last judgement we have to define. The corresponding rules use substitutions of a type B for a variable X in a type A, denoted by $A\{B/X\}$, which is defined as expected. All rules involving initial or final dialgebras have as side-condition that the corresponding types are well-formed.

$$\frac{\Delta \mid \Gamma \vdash A : *}{\Delta \mid \Gamma, x : A \vdash x : A}\ \textbf{(Proj)} \qquad \frac{\Delta \mid \Gamma_1 \vdash t : A \qquad g : \Gamma_2 \to \Gamma_1}{\Delta \mid \Gamma_2 \vdash t[g] : A[g]}\ \textbf{(Subst)}$$

$$\frac{}{\Gamma_k, x : A_k\{\mu(X; \overrightarrow{f}; \overrightarrow{A})/X\} \vdash \alpha_k : \mu(X; \overrightarrow{f}; \overrightarrow{A})[f_k]}\ \textbf{(Ind-I)}$$

$$\frac{}{\Gamma_k, x : \nu(X; \overrightarrow{f}; \overrightarrow{A})[f_k] \vdash \xi_k : A_k\{\nu(X; \overrightarrow{f}; \overrightarrow{A})/X\}}\ \textbf{(Coind-E)}$$

$$\frac{\Gamma \vdash C : * \qquad \Gamma_k, y : A_k\{C/X\} \vdash g_k : C[f_k]}{\Gamma, z : \mu(X; \overrightarrow{f}; \overrightarrow{A}) \vdash \mathrm{rec}\ \overrightarrow{g} : C}\ \textbf{(Ind-E)}$$

$$\frac{\Gamma \vdash C : * \qquad \Gamma_k, y : C[f_k] \vdash g_k : A_k\{C/X\}}{\Gamma, z : C \vdash \mathrm{corec}\ \overrightarrow{g} : \nu(X; \overrightarrow{f}; \overrightarrow{A})}\ \textbf{(Coind-I)}$$

We see that the domain of the constructors α_k for $\mu(X; \overrightarrow{f}; \overrightarrow{A})$ is determined by A_k and their codomain by substituting along f_k. Dually, the domain of destructors ξ_k is given by substituting f_k and their codomain by A_k. Note also that the bound variables in (co)recursions are implicit.

This concludes the definition of our proposed calculus. Note that there are no primitive type constructors for $\to$-, Π- or Σ-types, all of these are, together with the corresponding (weak) introductions and eliminations, definable in the above calculus. We will see this in the talk.

As basic example, assuming we have already defined the singleton type $\mathbf{1}$ and binary products, then we can define vectors $\Gamma \vdash \mathrm{Vec}\, A : *$ in context $\Gamma = n : \mathbb{N}$ by

$$\begin{array}{ll} \mathrm{Vec}\, A = \mu(X; (f_1, f_2); (\mathbf{1}, A \times X[k])) & \\ \Gamma_1 = \emptyset & \Gamma_2 = k : \mathbb{N} \\ f_1 = (0) : \Gamma_1 \to \Gamma & (\Gamma \vdash X : *) \mid \Gamma_1 \vdash \mathbf{1} : * \\ f_2 = (k+1) : \Gamma_2 \to \Gamma & (\Gamma \vdash X : *) \mid \Gamma_2 \vdash A \times X[k] : * \end{array}$$

This yields the constructors $x : \mathbf{1} \vdash \alpha_1 : \mathrm{Vec}\, A[0]$ and $k : \mathbb{N}, x : A \times \mathrm{Vec}\, A[k] \vdash \alpha_2 : \mathrm{Vec}\, A[k+1]$, which are usually called nil and cons.

In the talk, we show that the above calculus has indeed the structure we are aiming for, in the sense that types in context give rise to a split fibration, if we employ the expected equations on substitutions ($A[\mathrm{id}] = A$, $x_i[f] = f_i$, etc.). This allows us to conveniently define actions of types on terms, and a reduction relation. For now, we have no proof of type preservation etc.

References

[APTS13] Andreas Abel, Brigitte Pientka, David Thibodeau, and Anton Setzer. Copatterns: Programming infinite structures by observations. In *Proc. of POPL*, pages 27–38. ACM, 2013.

[Hag87] Tatsuya Hagino. A typed lambda calculus with categorical type constructors. In *Category Theory in Computer Science*, pages 140–157, 1987.

[HF11] Makoto Hamana and Marcelo Fiore. A foundation for GADTs and inductive families: Dependent polynomial functor approach. In *Proceedings of WGP'11*, pages 59–70. ACM, 2011.