Impredicative Encodings of (Co)inductive Types

Steven Bronsveld[1], Herman Geuvers[12], and Niels van der Weide[1]

[1] iCIS, Radboud University Nijmegen, The Netherlands
[2] Technical University Eindhoven, The Netherlands

Abstract

In impredicative type theory (System F, also known as $\lambda 2$, [3, 4]), it is possible to define inductive data types, such as natural numbers. It is also possible to define coinductive data types such as streams. Their (co)recursion principles obey the expected computation rules (the β-rules), but unfortunately, they do not yield a (co)induction principle [2, 6], because the necessary uniqueness principles are missing (the η-rules). Awodey, Frey, and Speight [1] used a extension of the Calculus of Constructions with Σ-types, identity-types, and functional extensionality to define System F style inductive types with an induction principle, by encoding them as a well-chosen subtype, making them initial algebras. We extend their results to coinductive data types, and below we detail the stream data type with the desired coinduction principle (also called bisimulation). To do that, we first define quotient types (with the desired η-rules) and we also need a stronger form of the definable existential types. The method of [1] can be used for general inductive types by defining W-types with an induction principle. The dual approach for streams can be extended to M-types, the generic notion of coinductive types, and the dual of W-types.

To define final coalgebras impredicatively, we first need quotient types. The well-known impredicative definition of quotient types satisfies the β-rule, but fails to satisfy the η-rule. We extend the technique of [1] to define quotients with an η-rule. We fix a type $D : \mathcal{U}$ and a relation $R : D \to D \to \mathcal{U}$. We say that a function f **respects** R (as a type: $\mathtt{EqCls}\ f\ R$) if for all $x, y : D$ such that $R\ x\ y$, we have $f\ x = f\ y$. The well-known impredicative **quotient type**, $\mathtt{quot}^*\ D\ R$, is defined as $\Pi(C : \mathcal{U}).\Pi(f : D \to C).\mathtt{EqCls}\ f\ R \to C$. The **class function** $\mathtt{cls}^* : D \to \mathtt{quot}^*\ D\ R$ is defined to be

$$\mathtt{cls}^* := \lambda(d : D).\lambda(C : \mathcal{U}).\lambda(f : D \to C).\lambda(H : \mathtt{EqCls}\ f\ R).f\ d.$$

We can lift a function $(f : D \to E)$ that respects R to a function $(\overline{f} : \mathtt{quot}^*\ D\ R \to E)$. This lifting is done by the **recursor for quotient types**, $\mathtt{rec}^*_\mathtt{q}$:

$$\mathtt{rec}^*_\mathtt{q} := \lambda(C : \mathcal{U}).\lambda(f : D \to C).\lambda(H : \mathtt{EqCls}\ f\ R).\lambda(q : \mathtt{quot}^*\ D\ R).q\ C\ f\ H$$

As usual we write $\overline{f}$ for $(\lambda q.\ \mathtt{rec}^*_\mathtt{q}\ C\ f\ H\ q)$. We have that $\overline{f} \circ \mathtt{cls}^* = f$, which gives the β-rule for quotients. NB. We do not require R to be an equivalence relation, so $\mathtt{quot}^* DR$ is actually the quotient of D by the smallest equivalence relation containing R.

The **inductive quotient type** (that will satisfy the η-rule), which we denote by $\mathtt{quot}$, is defined as $\Sigma(q : \mathtt{quot}^*\ D\ R).\mathtt{LimQuot}\ q$ where $\mathtt{LimQuot}\ (q : \mathtt{quot}^*\ D\ R)$ says that for all functions $g : D \to X$ and $g' : D \to Y$ that respect R, and functions $f : X \to Y$, we have $f\ (\mathtt{rec}^*_\mathtt{q}\ X\ g\ H\ q) = \mathtt{rec}^*_\mathtt{q}\ Y\ g'\ H'\ q$ whenever $f \circ g = g'$. We define the **recursor** as follows.

$$\mathtt{rec}_\mathtt{q} := \lambda(C : U).\lambda(f : D \to C).\lambda(H : \mathtt{EqCls}\ f\ R).\lambda(q : \mathtt{quot}\ D\ R).\mathtt{rec}^*_\mathtt{q}\ C\ f\ H\ (\mathtt{pr1}\ q)$$

It can be shown that we have a term $\mathtt{LimQuotCls} : (\mathtt{LimQuot}\ (\mathtt{cls}\ d))$, and using that we define the **class function** $\mathtt{cls}$ to be $\lambda(d : D).\langle \mathtt{cls}^*\ d,\ \mathtt{LimQuotCls}\ d\rangle$. We can now show that the general η-rule is satisfied: If $(g : D \to C)$ respects R, then for every $(f : D/R \to C)$ with $f \circ \mathtt{cls} = g$, we have $f = \overline{g}$.

It is standard to define the **impredicative existential type** $\exists^*(X:\mathcal{U}).P(X)$ to be $\Pi(Y:\mathcal{U}).(\Pi(X:\mathcal{U}).(P\,X)\to Y)\to Y$. The constructor $\mathtt{pack}^*$ and recursor $\mathtt{rec}^*_\exists$ are defined to be

$$\begin{array}{rcl}\mathtt{pack}^* & := & \lambda(X:\mathcal{U}).\lambda(t:P\,X).\lambda(Y:\mathcal{U}).\lambda(k:\Pi(X:\mathcal{U}).(P\,X)\to Y).k\ X\ t\\ \mathtt{rec}^*_\exists & := & \lambda(Y:\mathcal{U}).\lambda(f:\Pi(Z:\mathcal{U}).(P\,Z)\to Y).\lambda(e:\exists^*X.P).e\ Y\ f\end{array}$$

These do not satisfy the desired equations: when we *unpack* an $(e:\exists X.P)$ using the recursor $\mathtt{rec}_\exists$, obtaining $(X:\mathcal{U})$ and $(t:P(X))$, and then we re-package them into $e' := \mathtt{pack}\ X\ t$, we want $e = e'$. We can define an improved existential type that satisfies the properties that we need: For all $e:\exists X.P$ there are $X:\mathcal{U}$ and $t:P\,X$ such that $e = \mathtt{pack}\,X\,t$. In addition, $\mathtt{rec}_\exists\ (\exists X.P)\ \mathtt{pack} = \mathtt{id}_{\exists X.P}$.

The well-known **impredicative stream type** $\mathtt{Stream}^*$ (over a carrier type E) is defined as $\exists(X:\mathcal{U}).X\times(X\to E)\times(X\to X)$. The destructors $\mathtt{hd}^*:\mathtt{Stream}^*\to E)$ and $\mathtt{tl}^*:\mathtt{Stream}^*\to\mathtt{Stream}^*$ are standard to define, and so is the **corecursor** (that produces a stream):

$$\mathtt{corec}^*_{\mathtt{s}} := \lambda(X:\mathcal{U}).\lambda(h:X\to E).\lambda(t:X\to X).\lambda(x:X).\mathtt{pack}\ X\ \langle x,h,t\rangle.$$

It is a simple check that the introduction rule (corecursor) and elemination rules (head and tail functions) compute as one would expect. We want to define a stream-type that satisfies the η-rule, which is taken from the final coalgebra: Given a stream-morphism $(f:X\to Y)$, we want that $u_Y\circ f = u_X$ for the morphisms $u_X := \mathtt{corec}^*_{\mathtt{s}}\ X\ h\ t$ and $u_Y := \mathtt{corec}^*_{\mathtt{s}}\ Y\ h'\ t'$.

In the case of quotients (and for natural numbers, see [1]), we create a *subtype* of $\mathtt{quot}^*\ D\ R$. In the case of streams, we take the dual notion: a *quotient*. To define this quotient, we define a relation $\mathtt{CoLimStr}$ that relates streams σ and τ if they can be translated into each other by some stream-morphism.

$$\begin{array}{rcl}\mathtt{CoLimStr}\ \sigma\ \tau & := & \exists(X,Y:\mathcal{U}).\exists^{(h:X\to E)}_{(h':Y\to E)}.\exists^{(t:X\to X)}_{(t':Y\to Y)}.\exists(f:X\to Y).\exists(x:X)\\ & & (\mathtt{MorphStream}\ X\ h\ t\quad Y\ h'\ t'\quad f)\ \wedge\\ & & \sigma = \mathtt{corec}^*_{\mathtt{s}}\ X\ h\ t\ x\ \wedge\ \tau = \mathtt{corec}^*_{\mathtt{s}}\ Y\ h'\ t'\ (f\ x)\end{array}$$

We use the infix notation $(\sigma\ \equiv\ \tau)$ to denote that $(\mathtt{CoLimStr}\ \sigma\ \tau)$ holds. The relation $\mathtt{CoLimStr}$ is neither *symmetric* nor *transitive*. This does not hinder us since the equality relation on the quotient is an equivalence relation and we will have $\mathtt{cls}\,\sigma = \mathtt{cls}\,\tau$ if $(\mathtt{CoLimStr}\ \sigma\ \tau)$ holds. The **coinductive stream type** $\mathtt{Stream}$ is defined to be $\mathtt{quot}\ \mathtt{Stream}^*\ \mathtt{CoLimStr}$. To define the new head and tail functions as lifted functions of $\mathtt{hd}^*$ and $\mathtt{tl}^*$ we need to show that $\mathtt{hd}^*$ and $\mathtt{tl}^*$ respect $\equiv$. Using that we define the destructors for the new stream type: $\mathtt{hd}:\mathtt{Stream}\to E$ and $\mathtt{tl}:\mathtt{Stream}\to\mathtt{Stream}$. Finally, we define the corecursor $\mathtt{corec}_{\mathtt{s}}:\Pi(X:\mathcal{U}).(X\to E)\to(X\to X)$ to be $\mathtt{cls}\circ\mathtt{corec}^*_{\mathtt{s}}$. Using this we can show the η-rule for $\mathtt{Stream}^*$: For all $(X:\mathcal{U}),(h:X\to E),(t:X\to X)$ and $(f:X\to\mathtt{Stream})$, if $(\mathtt{MorphStream}\ X\ h\ t\ \mathtt{Stream}\ \mathtt{hd}\ \mathtt{tl}\ f)$, then we have $(f = \mathtt{corec}_{\mathtt{s}}\ X\ h\ t)$.

Two streams σ,τ are bisimilar, $\sigma\sim\tau$, if there is a bisimulation relation that relates them, where R is a **bisimilation** if for all $\sigma,\tau:\mathtt{Stream}$ we have that $\mathtt{hd}\ \sigma = \mathtt{hd}\ \tau$ and $R\ (\mathtt{tl}\ \sigma)\ (\mathtt{tl}\ \tau)$ whenever $R\ \sigma\ \tau$. We have the *coinduction* proof principle, stating that bisimilarity and equality coincide. More precisely: for all $\sigma,\tau:\mathtt{Stream}$, we have $\sigma\sim\tau$ if and only if $\sigma=\tau$.

We used the limit predicate to encode the inductive data types, similar to [1]. For coinductive data types, we dually used the colimit. It is also possible to directly encode the induction principle within the embedding, by creating a predicate $\mathtt{Ind}$ as was done in [5]. Similarly for coinductive types, one can use the $\mathtt{CoInd}$ principle by quotienting with bisimilarity relation. It is also possible to encode the η-rule directly by representing the uniqueness requirement, using a predicate $\mathtt{Unq}$.

References

[1] Steve Awodey, Jonas Frey, and Sam Speight. Impredicative encodings of (higher) inductive types. In *Proceedings of the 33rd Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2018, Oxford, UK, July 09-12, 2018*, pages 76–85. ACM, 2018.

[2] Herman Geuvers. Induction is not derivable in second order dependent type theory. In Samson Abramsky, editor, *Typed Lambda Calculi and Applications, 5th International Conference, TLCA 2001, Krakow, Poland*, volume 2044 of *Lecture Notes in Computer Science*, pages 166–181. Springer, 2001.

[3] Jean-Yves Girard, Paul Taylor, and Yves Lafont. *Proofs and types*. Cambridge University Press, 1993.

[4] John C. Reynolds. Towards a theory of type structure. In *Programming Symposium, Proceedings Colloque sur la Programmation, Paris, France, April 9-11, 1974*, volume 19 of *Lecture Notes in Computer Science*, pages 408–423. Springer, 1974.

[5] Xavier Ripoll Echeveste. Alternative impredicative encodings of inductive types. Master's thesis, Universiteit van Amsterdam, 2023.

[6] Ivar Rummelhoff. Polynat in PER models. *Theor. Comput. Sci.*, 316(1):215–224, 2004.