(Co)datatypes via W-setoids and M-setoids

Galaad Langlois, Damien Pous, and Yannick Zakowski

LIP — CNRS, Inria, ENS de Lyon

Motivation and objectives. Coq's native support for inductive and coinductive types is nice but suffers from a few limitations: (1) the default equality (Leibniz) on coinductive types is too strong, (2) the positivity condition for defining such types is syntactic, (3) the guard condition for defining fixpoints and cofixpoints is also syntactic. Point (1) usually forces us to define an appropriate notion of bisimilarity for each new coinductive type. Point (2) prevents us from defining inductive and coinductive types in a compositional way, and forces us to use encodings to obtain mixed inductive-coinductive types. Point (3) is not so problematic for inductive types, since one may always resort to well-founded recursion, but is almost a showstopper when it comes to coinductive types [5, 10]. We propose a formalisation of setoid variants of W-types and M-types in order to circumvent these limitations [7].

While not as general as Coq's native inductive and coinductive types, we claim that such an approach could prove useful for many datatypes. For instance, eventually, this will make it possible to easily define and work with datatypes such as

1. $\mu X.\ A \times \mathcal{F}(X)$: finite extensional trees labelled in A. Here the μ notation denotes the construction of an inductive type, and $\mathcal{F}(X)$ is an encoding for finite sets of elements of X. Such an inductive type is not allowed natively since $\mathcal{F}$ does not satisfy the syntactic positivity condition (a priori).
2. $\nu X.\ A \times \mathcal{F}(X)$: possibly infinite but finitely branching extensional trees labelled in A. Here the ν notation denotes the construction of a coinductive type, and the occurrence of $\mathcal{F}$ yields the same difficulty as above.
3. $\nu X.\mu Y.(Y + X)$: the mixed coinductive-inductive type of 'clocks', i.e., streams of zeros and ones with infinitely many ones.

Inspiration. Category theory gives a solid starting point: inductive (resp. coinductive) types can be seen as initial algebras (resp. final coalgebras) of appropriate functors [6]. However, not all functors admit initial algebras and final coalgebras.

In Isabelle/HOL, Blanchette et al. give a solution based on *bounded natural functors* (BNF), which they implemented in the AmiCo library [3]. Such functors are closed under various useful constructions, and they have initial algebras and final coalgebras. Moreover, using the concept of 'friend', they make it possible to define functions by (co)recursion up-to. Such a formalisation however relies on powerful features of the host logic (HOL). E.g., the axiom of choice.

In Lean, Avigad et al. [2] proposed the use of *quotients of polynomial functors* (QPF), which actually generalise bounded natural functors. Like in Isabelle/HOL, this path however requires the axiom of choice, via the use of quotient types (together with propositional extensionality and functional extensionality).

In order to implement these structures in Coq while remaining axiom-free, we build them in the category of setoids, i.e. types equipped with an equivalence relation. This setup makes it natural to solve issue (1): we equip coinductive datatypes directly with the right notion of equality (i.e., bisimilarity).

Construction on paper. We focus on univariate polynomial functors, which are the key ingredient. Those are sometimes called *containers*; we follow the presentation from [1].

A container $(A \triangleright B)$ is a pair of a domain of shapes `A : Type` and a type family `B : A → Type` which associates to each shape a domain of indices. To each container $(A \triangleright B)$ is associated a so-called polynomial functor (also designed as the extension of the container), defined as $P(X) = \sum_{a:A}(B(a) \to X)$. Containers provide a useful interface as they are closed under product, sum and composition. Every container admits an initial algebra (resp. final coalgebra) defined as its so-called W-type (resp. M-type). W-types (resp. M-types) are built as the inductive (resp. coinductive) tree where each node stores a shape `a : A`, and the branches from that node are indexed by `B a`.

Axiom-free implementation: the need for setoids. Without assuming functional extensionality, one cannot prove the initiality of W-types. This issue is solved by defining instead *W-setoids*, i.e. by equipping W-types with a suitable equivalence relation. For M-types, the need for setoids is more obvious: bisimilarity is the right notion of equality for them, and it does not coincide with Leibniz equality. Consequently, we define *M-setoids* as M-types associated with a bisimularity relation (implemented via the coinduction package [10]).

Categorically, this means that the objects we consider in the category are no more types but setoids, and the arrows are no more functions but extensional functions, i.e. functions that respect setoid equivalence. W-setoids (resp. M-setoids) are initial algebras (resp. final coalgebras) of polynomial functors of setoids. In order for everything to work smoothly, especially for the composition of containers, we move to the setoid counterpart to containers, as described in [4]. Those are pairs of a setoid domain of shapes `A : Setoid` and a *setoid family* `B : A → Setoid`. Such families are more demanding than in the category of types: given two equivalent shapes a and a' in A, one should be able to translate elements of $B(a)$ into elements of $B(a')$, and vice-versa, in a proof-irrelevant way [8].

At the time of submission, we have a toolbox for defining and composing polynomial functors, computing their containers, and producing their initial algebras and final coalgebras. To this end, we have defined W-setoids and proved that the W-setoid of a container is indeed its initial algebra (a proof already present in [9, 4]). Similarly, we have defined M-setoids and proved that they yield final coalgebras.

Future work. As introduced in [1] and implemented in Lean by Avigad et al. [2] (in a category of types which is extensional thanks to strong axioms), the natural next step is to handle multi-parameters containers, i.e., multivariate polynomial functors. This extension allows for defining parameterised inductive types; for instance, `list X` is the initial algebra over Y of the bivariate functor $\mathcal{L}(X, Y) = 1 + X \times Y$. A key property is that multivariate polynomial functors are closed under initial algebra and final coalgebra constructions; thus `list` as defined above is itself a (univariate) polynomial functor. This is also how one can deal with inductive-coinductive datatypes as in the third example given above: $\lambda X.\mu Y.(Y + X)$ is a polynomial functor.

Moving to quotients of polynomial functors [2] brings another orthogonal gain in expressiveness. For instance, while the functor $\mathcal{F}$ used in the first two examples above is not polynomial, it is a quotient of the list functor $\mathcal{L}$ (considering concatenation modulo associativity, commutativity, and idempotence).

Once this has been achieved, the next step will consist in adding support for (co)recursion up to friends as done in Isabelle/HOL AmiCo's package [3], to solve issue (3).

References

[1] M. Abbott, T. Altenkirch, and N. Ghani. Categories of containers. In *Foundations of Software Science and Computation Structures*, pages 23–38. Springer, 2003.

[2] J. Avigad, M. Carneiro, and S. Hudon. Data types as quotients of polynomial functors. In *ITP*, volume 141 of *LIPIcs*, pages 6:1–6:19. Schloss Dagstuhl, 2019.

[3] J. C. Blanchette, A. Bouzy, A. Lochbihler, A. Popescu, and D. Traytel. Friends with benefits - implementing corecursion in foundational proof assistants. In *ESOP*, volume 10201 of *LNCS*, pages 111–140. Springer, 2017.

[4] J. Emmenegger. W-types in setoids. *Logical Methods in Computer Science*, 17, 2021.

[5] C. Hur, G. Neis, D. Dreyer, and V. Vafeiadis. The power of parameterization in coinductive proof. In *POPL*, pages 193–206. ACM, 2013.

[6] B. Jacobs. *Introduction to Coalgebra*, volume 59. Cambridge University Press, 2016.

[7] G. Langlois, D. Pous, and Y. Zakowski. Accompanying code. http://perso.ens-lyon.fr/galaad.langlois/final-coalgebras.zip, 2023.

[8] E. Palmgren. Proof-relevance of families of setoids and identity in type theory. *Archive for mathematical logic*, 51(1-2):35–47, 2012.

[9] E. Palmgren. Wsetoid.v. https://staff.math.su.se/palmgren/coq/czf_and_setoids/Wsetoid.v, 2015.

[10] D. Pous. Coinduction all the way up. In *LICS*, pages 307–316. ACM, 2016. Associated library: coq-coinduction, available via opam or at https://github.com/damien-pous/coinduction.