Comonadic Iteration*

Tarmo Uustalu[†‡]
Inst. of Cybernetics, Tallinn Tech. Univ.
Akadeemia tee 21, EE-12618 Tallinn, Estonia; `tarmo@cs.ioc.ee`

Varmo Vene[†]
Inst. of Computer Science, Univ. of Tartu
J. Liivi 2, EE-50409 Tartu, Estonia; `varmo@cs.ut.ee`

Alberto Pardo
Instituto de Computación, Universidad de la República
Julio Herrera y Reissig 565, 11300 Montevideo, Uruguay; `pardo@fing.edu.uy`

The aim of generic/polytypic programming is to devise programming languages and calculi supporting non-traditional forms of polymorphism, the prime example in the functional setting being polymorphism in type constructors. In this paper, we discuss generic *recursion combinators*, i.e., combinator codings of recursive function definition schemes working uniformly on any inductive source type of functions. Such combinators are polymorphic in a type constructor inducing an inductive type. Generic recursion schemes are numerous. The most fundamental and easy to code by a combinator are (simple) iteration and primitive-recursion. Some other schemes that are more general in terms of direct expression power, e.g., the course-of-value strengthenings of iteration and primitive-recursion, are equally useful in programming and do not lead to considerably higher sophistication in the formulation [1]. Too particular generalizations, however, get too esoteric and sophisticated. To make a library primitive out of every powerful imaginable recursion scheme is therefore not a good idea. It might be the case that a better approach is to add another dimension of genericity. In this paper, we suggest a "many-in-one" recursion combinator that neatly captures a whole family of recursion schemes that prima facie present themselves as diverging generalizations of simple iteration. The combinator is doubly generic: in addition to being polymorphic in a signature for an inductive type, it is polymorphic in a *comonad* and a natural transformation, which together determine a particular recursion scheme. The family includes simple primitive-recursion, course-of-value versions of iteration and primitive-recursion, and much more. Below is a brief mathematical summary.

Let $\mathcal{C}$ be a category and $F : \mathcal{C} \to \mathcal{C}$ be a functor. An *F-algebra* is a morphism $\phi : F\,C \to C$, where C (its carrier) is an object of $\mathcal{C}$. If ψ, ϕ are two F-algebras with carriers D, C, then an *F-algebra map* from ψ to ϕ is a morphism $h : D \to C$ such that $f \circ \psi = \phi \circ F\,f$. An initial F-algebra (if existing) is therefore a morphism $\mathsf{in}_F : F\,\mu F \to \mu F$ with the property that, for any morphism $\phi : F\,C \to C$, there is exactly one $f = (\!|\, \phi \,|\!)_F : \mu F \to C$ (the *catamorphism* or *fold* given by ϕ) satisfying $f \circ \mathsf{in}_F = \phi \circ F\,f$. The carrier μF models the inductive type with signature F; the algebra operation in_F models the associating constructors. $(\!|\, \cdot \,|\!)_F$ captures iteration, or (simple) structural recursion, the simplest scheme of defining functions on μF recursively.

Suppose now that $N = (N, \varepsilon, \delta)$ is a comonad on $\mathcal{C}$. This means that $N : \mathcal{C} \to \mathcal{C}$ is a functor and $\varepsilon : N \to \mathsf{Id}$ and $\delta : N \to NN$ are two natural transformations such that $\varepsilon_{N\,A} \circ \delta_A = N\,\varepsilon_A \circ \delta_A = \mathsf{id}_{N\,A}$ and $\delta_{N\,A} \circ \delta_A = N\,\delta_A \circ \delta_A$ for any object A of $\mathcal{C}$. Suppose, further, that $\kappa : F\,N \to N\,F$ is a natural transformation such that $\varepsilon_{F\,A} \circ \kappa_A = F\,\varepsilon_A$ and $\delta_{F\,A} \circ \kappa_A = N\,\kappa_A \circ \kappa_{N\,A} \circ F\,\delta_A$. Then, for any morphism $\phi : F\,N\,C \to C$, there turns out to be a unique $f = (\!|\, \phi \,|\!)_F^{N,\kappa} : \mu F \to C$ (a *comonadic catamorphism*) such that

$$f \circ \mathsf{in}_F = \phi \circ F\,(N\,f \circ (\!|\, N\,\mathsf{in}_F \circ \kappa_{\mu F} \,|\!)_F)$$

*Presented at the 12th Nordic Workshop on Programming Theory, October 11–13, 2000, Bergen, Norway, http://www.ii.uib.no/~n wpt00

†On leave to Dept. of Inform., Univ. of Minho, Campus de Gualtar, P-4710-057 Braga, Portugal; `tarmo@di.uminho.pt`. Supported by the Portuguese Foundation for Science and Technology under grant no. PRAXIS XXI/C/EEI/14172/98.

‡Partially supported by the Estonian Science Foundation under grant no. 4155.

A comonadic catamorphism is generally not a catamorphism, but is always definable in terms of one: if $\phi : F\,N\,C \to C$, then $(\!|\,\phi\,|\!)_F^{N,\kappa} = \varepsilon_C \circ (\!|\,N\,\phi \circ \kappa_{N\,C} \circ F\,\delta_C\,|\!)_F$. On the other hand, a catamorphism is always a comonadic catamorphism: if $\phi : F\,C \to C$, then $(\!|\,\phi\,|\!)_F = (\!|\,\phi \circ F\,\varepsilon_C\,|\!)_F^{N,\kappa}$. Like catamorphisms, comonadic catamorphisms enjoy a number of neat calculational properties.

By comonadic iteration, we mean the "very generic" recursion scheme captured by the comonadic catamorphism combinator. This scheme is a unification of a number of particular generic and specific recursion schemes. The product comonad, for example, gives us the asymmetric form of mutual iteration (zygomorphisms, [2]), including the most important special case, primitive-recursion (paramorphisms, [3]). The generalized stream comonad gives course-of-value iteration (histomorphisms, [1]) and an interesting generalization permitting, for instance, recursive calls of the definiendum on "rearrangements" of recursive components of the given argument value. Another type of comonad giving rise to meaningful recursion schemes is that of the state-in-context comonad. In the paper, we illustrate the theory by a Haskell implementation of the combinators and usage examples discussed.

Comonadic iteration in the sense of this paper is not the only possible combination of iteration and comonads. Pardo [4] discusses a different merger. He proposes a combinator that takes morphisms $\phi : N\,F\,C \to C$ to morphisms $\mathsf{cfold}_F^{N,\theta}(\phi) : N\,\mu F \to C$. This requires F and N to be related by a natural transformation $\theta : N\,F \to F\,N$. In the case of the product comonad, the combinator gives him to forms of iteration with extra parameters, one with fixed parameters (pfolds) and the other with accumulating parameters (afolds). It is the hope of the authors that the two orthogonal approaches admit a fruitful combination.

A step in the direction of our comonadic iteration was taken by Lenisa [5]. (As a matter of fact, she looked into monadic coiteration, but this is dual to comonadic iteration.) She recorded the obvious fact that any morphism $\phi' : F\,N\,C \to N\,C$ determines a morphism $f : \mu F \to C$, namely, $f = \varepsilon_C \circ (\!|\,\phi'\,|\!)_F$, but did not introduce κ. Without this, however, one cannot give the explicit definition and characterization for f in terms of $\phi = \varepsilon_C \circ \phi' : F\,N\,C \to C$ and turn comonadic iteration into a true recursion scheme.

References

[1] T Uustalu and V Vene. Primitive (co)recursion and course-of-value (co)iteration, categorically. *INFORMATICA*, 10(1):5–26, 1999.

[2] G R Malcolm. Algebraic data types and program transformation. PhD thesis, Dept of Computer Science, Univ of Groningen, 1990.

[3] L Meertens. Paramorphisms. *Formal Aspects of Computing*, 4(5):413–424, 1992.

[4] A Pardo. Towards merging recursion and comonads. In J Jeuring, ed, *Proceedings 2nd Workshop on Generic Programming, WGP'2000, Ponte de Lima, Portugal, 6 July 2000*, Tech Report UU-CS-2000-19, Dept of Computer Science, Utrecht Univ, pp 50–68. June 2000.

[5] M Lenisa. From set-theoretic coinduction to algebraic coinduction: Some results, some problems. In B Jacobs and J Rutten, eds, *Proceedings 2nd Int Workshop on Coalgebraic Methods in Computer Science, CMCS'99, Amsterdam, The Netherlands, 20–21 March 1999*, v 19 of *Electronic Notes in Theoretical Computer Science*. Elsevier, North-Holland, 1999.