The Delay Monad and Restriction Categories

James Chapman, Tarmo Uustalu and Niccolò Veltri

Institute of Cybernetics, Tallinn University of Technology, Tallinn, Estonia
{james,tarmo,niccolo}@cs.ioc.ee

Restriction categories are an abstract axiomatic framework by Cockett and Lack for reasoning about partiality of functions [3]. In a restriction category, every map $f : A \to B$ is required to define an endomap $\overline{f} : A \to A$, satisfying four equational conditions:

$$\begin{aligned} &R1. \quad f \circ \overline{f} = f \\ &R2. \quad \overline{g} \circ \overline{f} = \overline{f} \circ \overline{g} \\ &R3. \quad \overline{g} \circ \overline{f} = \overline{g \circ \overline{f}} \\ &R4. \quad \overline{g} \circ f = f \circ \overline{g \circ f} \end{aligned}$$

A map f is called total, if $\overline{f} = \mathrm{id}$. The map $\overline{f}$ should be thought of as a partial identity function on A specifying the domain of definedness of f. The restriction operator also defines a partial order on maps, $f \leq g$ if and only if $f = g \circ \overline{f}$. That is, f is less defined than g if f coincides with g on f's domain of definedness.

Restriction categories are related to partial map categories, which are the classical synthetic approach to partiality: every partial map category is a restriction category and a restriction category is a partial map category intuitively whenever the restriction category contains as objects all the domains of definedness. Formally this is when all restriction idempotents split.

An example of a restriction category particularly relevant for dependently type programming is the Kleisli category of the delay monad. The delay monad was introduced by Capretta [2] as a means to incorporate general recursion to type theory and it is useful in this setting for modeling non-terminating behaviours. It constitutes a constructive alternative to the maybe monad. For a given type A, each element of $\mathsf{Delay}\ A$ is a possibly infinite computation that returns a value of A, if it terminates. We define $\mathsf{Delay}\ A$ as a coinductive type by the rules

$$\frac{}{\mathsf{now}\ a : \mathsf{Delay}\ A} \qquad \frac{c : \mathsf{Delay}\ A}{\mathsf{later}\ c : \mathsf{Delay}\ A}$$

Propositional equality is not suitable for coinductive types. We need different notions of equality, namely strong and weak bisimilarity. Two computations are strongly bisimilar, if they contain the same number of applications of later, i.e., it takes the same (possibly infinite) amount of time for them to converge to the same value. Strong bisimilarity is defined coinductively by the rules

$$\frac{}{\mathsf{now}\ a \sim \mathsf{now}\ a} \qquad \frac{c \sim c'}{\mathsf{later}\ c \sim \mathsf{later}\ c'}$$

Weak bisimilarity is defined in terms of convergence. This binary relation between $\mathsf{Delay}\ A$ and A relates a terminating computation to its value and is inductively defined by the rules

$$\frac{}{\mathsf{now}\ a \downarrow a} \qquad \frac{c \downarrow a}{\mathsf{later}\ c \downarrow a}$$

Two computations are weakly bisimilar if they differ by a finite number of application of the constructor later, i.e., they either converge to the same value or diverge. Weak bisimilarity is defined coinductively by the rules

$$\frac{c \downarrow a \quad c' \downarrow a}{c \approx c'} \qquad \frac{c \approx c'}{\mathsf{later}\ c \approx \mathsf{later}\ c'}$$

The functor Delay quotiented by strong/weak bisimilarity is a strong monad. The Kleisli category of the delay monad quotiented by weak bisimilarity ($\mathbf{Kl}(\mathsf{Delay}/\approx)$) is a restriction category. The restriction $\overline{f} : A \to \mathsf{Delay}\ A$ of a map $f : A \to \mathsf{Delay}\ B$ is given in terms of the strength σ of Delay by

$$\overline{f} = \ A \xrightarrow{\langle \mathrm{id}, f \rangle} A \times \mathsf{Delay}\ B \xrightarrow{\sigma_{A,B}} \mathsf{Delay}\ (A \times B) \xrightarrow{\mathsf{Delay}\ \pi_0} \mathsf{Delay}\ A$$

The restriction category $\mathbf{Kl}(\mathsf{Delay}/\approx)$ has a rich structure that makes it suitable for analyzing computability. It is a restriction category with a partial final object and partial binary products, with joins of compatible maps, meets of maps with semidecidable codomains and a uniform iteration operator.

The partial final object is 1 and the unique good map from an object A into 1 is $\mathsf{now} \circ !_A$. The partial product of A and B is $A \times B$ with projections $\mathsf{now} \circ \pi_0$ and $\mathsf{now} \circ \pi_1$. The pairing operation is defined on computations first and then extended pointwise to maps. It runs the two computations (e.g., sequentially) and returns a pair of values a, b when both computations have terminated with values a and b respectively.

The joins and meets in $\mathbf{Kl}(\mathsf{Delay}/\approx)$ are given by pointwise extensions to maps of joins and meets of compatible computations and computations over semidecidable types. Intuitively one runs the two given computations in parallel. The join of two computations returns the value of the quicker one of them (if at least one of the two computations terminates at all), while the meet terminates only when both have terminated, provided they gave the same value, else it goes on forever.

The category $\mathbf{Kl}(\mathsf{Delay}/\approx)$ also supports a uniform iteration operator $\mathsf{iter} : (A \to \mathsf{Delay}\ (A + B)) \to A \to \mathsf{Delay}\ B$. Informally, iteration is defined as follows. We apply a given map f to an initial value a. When $f\ a$ converges to an element of B, iteration has converged to that element. When it converges to an element of A, it is time to apply f again to that element.

We have formalized all of the development above in the dependently typed programming language Agda [1].

We are interested in learning more about the structure of $\mathbf{Kl}(\mathsf{Delay}/\approx)$. In particular we would like to show that it is a Turing category.

Acknowledgement This research was supported by the ERDF funded Estonian ICT national programme project "Coinduction", the Estonian Science Foundation grants No. 9219 and 9475 and the Estonian Ministry of Education and Research target-financed research theme No. 0140007s12.

References

[1] A. Bove, P. Dybjer, U. Norell. A brief overview of Agda, a functional language with dependent types. In *Proc. of TPHOLs 2009*, v. 5674 of *LNCS*, pp. 73–78. Springer, 2009.

[2] V. Capretta. General recursion via coinductive types. *Log. Meth. in Comput. Sci.*, v. 1, n. 2, article 1, 2005.

[3] J. R. B. Cockett and S. Lack. Restriction categories I: categories of partial maps. *Theor. Comput. Sci.*, v. 270, n. 1–2, pp. 223–259, 2002.