Monads in mathematics

Rémy Oudompheng

March 3, 2009

Abstract

This is an overview of the theory of monads and its applications.

Contents

1 Examples

1.1 Categories

Category theory studies in an abstract way how structures and constructions of mathematics are related. A *category* is a collection of (mathematical) objects. Usually, interesting categories contain objects sharing the same properties (there is a category of sets, a category of groups, a category of rings, and so on). A category need also have a definition of arrows, which often correspond to the usual definition of functions, maps or morphisms. But it is possible to define categories having more complicated arrows. A (not so) stupid non-trivial category is the opposite category C^{op} of a given category C, which has arrows going the other way.

1.2 Monads

A monad M on a category C is a *functor*: it associates to any object X in C another object MX of C in a so-called functorial way, which means that any arrow $X \to Y$ should give rise to an arrow $MX \to MY$. But in order to call M a monad, we require several other properties: there should be natural transformations $X \to MX$ (so that $X \to MX \to MY$ and $X \to Y \to MY$ are the same, which can be expressed by a *commutative square*), and $MMX \to MX$ such that $MX \to MMX \to MX$ is the identity (notice that there are two ways to obtain an arrow $MX \to MMX$). MacLane in *Categories for the Working Mathematician* [ML98] gives a good account of the theory along with a bit of history and references.

$$\begin{array}{ccc} X & \xrightarrow{\iota_X} & MX \\ {\scriptstyle f}\downarrow & & \downarrow{\scriptstyle Mf} \\ Y & \xrightarrow{\iota_Y} & MY \end{array} \qquad \begin{array}{ccc} MX & \xrightarrow{\iota_{MX}} & MMX \\ {\scriptstyle M(\iota_X)}\downarrow & & \downarrow \\ MMX & \longrightarrow & MX \end{array}$$

Most examples of monads arising in mathematics are derived from two concepts: operads, which combine monads with much richer structure (but correspond to really tangible examples), and adjunctions (a framework in which monads can actually fit). I should write about these later, and concentrate on examples. Monads are also used in computer science, because they model a construction scheme which is widely spread. The programming language *Haskell* formulates many concepts in the language of monads. Marco Maggesi wrote a small introduction to monads [Mag] (in Italian), which covers monads appearing in Haskell.

1.3 Basic examples

A basic undergraduate example is the monad of vector spaces. If k is a field, and X is a set, define $V_k(X)$ to be the vector space with basis X: this is the set of abstract linear combinations of elements of X with coefficients in k. Then V_k is a monad is the category of sets: it maps each set to another set (which is in fact a vector space over k), in a functorial way (any function $X \to Y$ defines a map of vector spaces $V_k(X) \to V_k(Y)$). We recognize the structure of a monad by considering the injection $X \to V_k(X)$ which maps an element of X to the corresponding basis vector. We also have a natural map $V_k(V_k(X)) \to V_k(X)$ which sends an abstract linear combination of elements of $V_k(X)$ to their *actual linear combination*, which is a well-defined element of $V_k(X)$. Checking the other properties of monads is a simple exercise.

In a similar way, it is possible to define a monad of associative monoids (which is also the `List` monad of Haskell), a monad of commutative monoids, a monad of rings, etc. These can all be expressed using the formalism of operads or of adjoint functors.

A more pictorial example of monad is the monad of trees, which is closely related to operads.

This monad associates to a set X the set TX of trees (to be defined) with a root (the bottom part of the tree in the picture) and leaves (the upper part) which are labelled by elements of the set (here a, b, c). The set X maps to TX: an element x is represented by the tree having no lines at all, just a point, labelled by x. The set TTX represents trees where leaves v carry a tree $T(v)$: such a tree-labelled tree has a natural associated tree, constructed by drawing really $T(v)$ at v, which makes a bigger tree (this operation is called *grafting trees*).

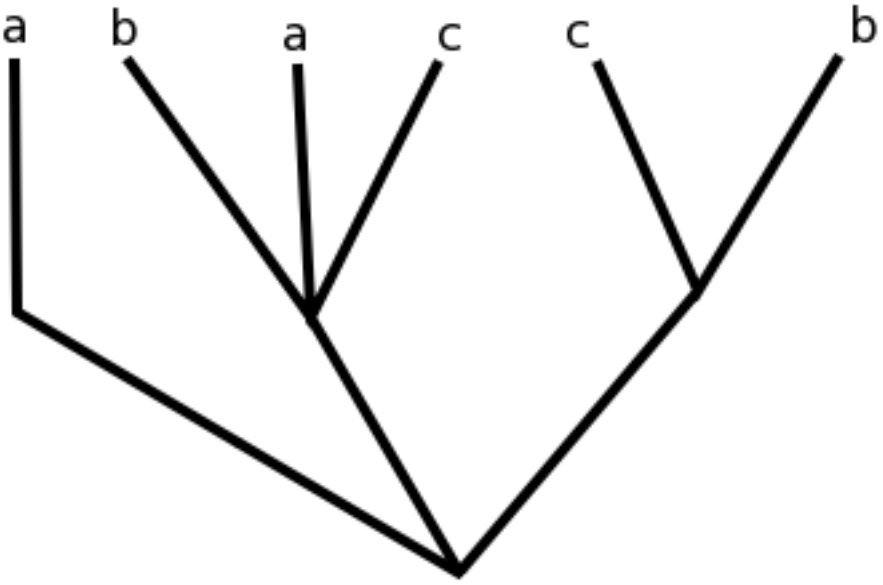


Figure 1: A tree with labelled leaves

2 Algebras over monads and operads

2.1 Monads from operads

A large class of monads is actually derived from operads: basic examples form monads in the category of sets and associate to a set X a set TX of abstract "terms" made of elements of X. For example, the monad of vector spaces I mentioned in the previous post is such a monad, and associates to a set the abstract linear combinations of its elements.

Operads are a generic structure giving a more precise definition of these terms. An operad is an abstract set of operations of various arities (an ugly word to precise the number or arguments taken by an operation: a ternary operation is said to have arity three), subject to relations between them or their compositions. For example, the operad of vector spaces consists of two basic operations: sum and product by scalars (which are actually infinitely many operations), which are tied by distributivity, commutativity and associativity among other relations. An example of operation of this monad is $(x, y, z) \to 2x + 3y + z$, which is a ternary operation.

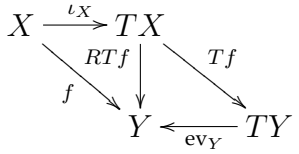
An operad T defines a natural associated monad, which associates to a set X the set TX of terms built on X, which is defined inductively by the following rules: atoms are the elements of X, and for any n-ary operation $o \in T$, and terms $t_1, \dots, t_n$, $o(t_1, t_2, \dots, t_n)$ is a term (several terms should be identified if the operad says they correspond to equivalent operations). Since ordinary definitions of operads already assume that it contains all compositions of operations, we usually only need t_i to be elements of X, which kills the need for recursion. However, this readily shows why T behaves like a monad: there is a natural map $X \to TX$, and the map $TTX \to TX$ comes from the fact that a term over TX is an operation taking terms of X as abstract inputs. It becomes a term over X by replacing these abstract inputs by real inputs and calculating the resulting composition.

2.2 Algebras over monads

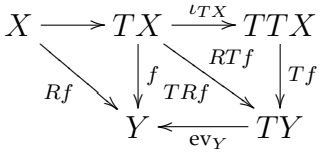
An algebra over a monad T is a set equipped with a morphism $TX \to X$, with the additional requirement that the associated morphism $TTX \to TX$ be exactly the composition of the monad T. In the case of an operad, this amounts to set a value in X for each term, that is, defining internal operations on X mimicking the properties of the abstract operations of the operad. For example, an algebra over the monad of vector space is exactly what we usually call a vector space.

We already know examples of algebras over an arbitrary monad T: since it comes with natural morphisms $TTX \to TX$, TX itself is always a T-algebra. We say that it is the free T-algebra over X. A free vector space over a set is the vector space having this set as a basis, a free commutative ring is a polynomial ring over indeterminates given by the base set, a free algebra over the monad/operad of groups is a free group, etc.

Let X be a set and $\mathrm{ev}_Y : TY \to Y$ be a T-algebra structure on a set Y. Any map $f : X \to Y$ defines naturally a map $Tf : TX \to Y$ which is $TX \to TY \to Y$. This corresponds to the fact that giving images for generators X produces maps from the free object TX to Y. Head-aching abstract nonsense proves actually that this correspondance is a bijection $\mathrm{Map}(X, \mathrm{Set}(Y)) = \mathrm{Hom}_T(TX, Y)$.



Indeed, a map $f : TX \to Y$ conversely restricts to $Rf : X \to Y$ defined on the generators. Given a map $f : X \to Y$, $f : X \to Y = (X \to Y \to TY) \to Y = (X \to TX \to TY) \to Y = RTf$. Given a map $f : TX \to Y$, we know that $Rf : X \to TX \to Y$ becomes $T(Rf) : TX \to TTX \to TY$ where $TTX \to TY = Tf$. Thus $TRf : TX \to TTX \to TY \to Y = RTf = f$.



We say that the forgetful functor sending a T-algebra Y to the ordinary set Y is (right) adjoint to the free T-algebra functor $X \to TX$.

3 Adjoint functors and general monads

3.1 Pairs of adjoint functors

The adjunction property between two functors, $T : C_1 \to C_2$ and $U : C_2 \to C_1$, says that there is a natural bijection between morphisms $\mathrm{Hom}_1(A, UB)$ (in the first category) and $\mathrm{Hom}_2(TA, B)$ (in the second category). Here *natural* means that these bijection is compatible with composition with morphisms $B \to B'$, $UB \to UB'$ or $A' \to A$ and $TA' \to TA$.

Adjunctions are naturally created by the use of monads or operads. For example, the functor $V_k : \mathrm{Set} \to \mathrm{Vect}_k$ mapping a set X to the free vector space $V_k(X) = k^{(X)}$ with basis X, has a adjoint, $U : \mathrm{Vect}_k \to \mathrm{Set}$, mapping a vector space to the set of its

elements. The meaning of the adjunction, is that a morphism $V_k(X) \to W$ is equivalent to the choice $X \to W$ of images of basis vectors where W is considered as as set. Similar adjunctions exist for other free objects (free algebras, free groups, free modules).

Another fundamental adjunction relates *products* and *exponentials* in the following sense. If X, Y and Z are sets, the bijection $\mathrm{Hom}(X \times Y, Z) = \mathrm{Hom}(X, Z^Y)$ adjoins the functors $\bullet \times Y$ and $\bullet^Y$. A similar adjunction works for modules or vector spaces, and states $\mathrm{Hom}(X \otimes Y, Z) \simeq \mathrm{Hom}(X, \mathrm{Hom}(Y, Z))$.

An informal way of describing adjoint functors is that one of the functors describes how to obtain morphisms to/from objects created by the other functor (these morphisms are said to be *representable*). For example, morphisms from a free vector space are described by picking elements in the target, so the "free vector space" functor is adjoint to the "set of elements" functor.

3.2 Monads from adjunction

Arbitrary pairs of adjoint functors define monads in the following way: if T and U are adjoint functors as before (T being the *left adjoint* and U the right adjoint). Recall that the adjunction between T and U says that U describes morphisms coming from objects of the form TX. Then the identity $TX \to TX$ should be represented by a morphism $X \to UTX$.

Is is then a simple but head-aching exercise to show that UT is a monad and understand what it means. When U is the "set of elements" functor and T the "free thing" functor, UT is the "free thing" monad in the sense of the first section. Beware that TU does not define a monad, but a comonad (whose definition has arrows going the other way): the natural morphism is $TUX \to X$.

The product/exponential adjunction defines a comonad $Y \times \mathrm{Hom}(Y, X) \to X$ which is the evaluation map $(y, f) \to f(y)$. There is also a monad $X \to \mathrm{Hom}(Y, X \times Y)$: applying it twice yields $\mathrm{Hom}(Y, \mathrm{Hom}(Y, X \times Y) \times Y)$ which has a natural morphism towards $\mathrm{Hom}(Y, X \times Y$ using the evaluation map. This monad can model computations with side effects (the `State` monad of Haskell) : these computations can usually be described as maps $S \times X_1 \to S \times X_2$, but these are exactly the same as maps $X_1 \to \mathrm{Hom}(S, S \times X_2)$, which syntaxically allows going from stateless objects to state-dependent objects.

4 The Kleisli category

5 Free operads

6 Resolutions with monads

References

[Bec03] Jonathan Mock Beck, *Triples, algebras and cohomology*, Repr. Theory Appl. Categ. **2** (2003), 1–59 (electronic).

[ML98] Saunders Mac Lane, *Categories for the working mathematician*, 2nd ed., Graduate Texts in Mathematics, vol. 5, Springer-Verlag, New York, 1998.

[Mag] Marco Maggesi, *Introduzione alla teoria delle monadi*, available at http://web.math.unifi.it/users/maggesi/monadi.pdf.