Monads? Arrows ? or Relative Monads ! for handling quantum effects

May 2025

Introduction

- Monads are fundamental in functional programming and category theory to model effects and substitution.
- Standard monads require endofunctors: $T : \mathcal{C} \to \mathcal{C}$.
- Many important constructions (e.g., syntax with variable binding, vector spaces) do not fit neatly into endofunctors.
- **Relative monads** generalize monads to functors $T : \mathcal{J} \to \mathcal{C}$ relative to a functor $J : \mathcal{J} \to \mathcal{C}$.

1 Superoperators as arrows

1.1 introduction

Monads (in)famously play an important role in functional programming. Monads capture notions of computation. With the prevalence of quantum computing in today's landscape, it is only sensible to ask whether monads can also describe quantum computations.

As part of the Adjoint School, we spent a lot of time in the last few months with this paper by Juliana K. Vozotto, Thorsten Altenkirch, and Amr Sabry which details how one can use the concept of arrows which generalize the notion of a monad, to model quantum computations. One question we encountered a lot while working with this paper is why exactly monads aren't strong enough to model quantum computation. Heuristically, this seems to be true, but this question deserves a concrete mathematical answer. In the following, we will highlight key concepts and constructions from the paper.

1.2 monads

For a mathematician, a monad infamously is just a monoid in the category of endofunctors. Explicitly, given a category $\mathcal{C}$, a monad consists of a functor

$T : \mathcal{C} \to \mathcal{C}$ together with natural transformations $\mu : T^2 \to T$ and $\eta : \mathrm{id}_{\mathcal{C}} \to T$ which need to satisfy associativity and unitality.

To a computer scientist, a monad consists of a type constructor T together with operations return : $a \to Ta$ and $\gg\!= : Ta \to (a \to Tb) \to Tb$, i.e., the monadic bind $\gg\!=$ takes a value of type Ta and a map $a \to Tb$ and returns a value of type Tb. These operations need to satisfy properties akin to associativity and unitality. Using currying, we can rewrite its signature as $\gg\!= : Ta \times \mathrm{Map}(a, Tb) \to Tb$. Monads are used in computer science to model partial functions, non-deterministic effects, exceptions, and states among many other examples.

These two notions of a monad do not seem to perfectly align at first. η and return do mirror each other. To see the correspondence between μ and $\gg\!=$, plugging id_{Ta} into the second component of $\gg\!=$ yields a map $T^2a \to Ta$ and, given a morphism $f : a \to Tb$, composing Tf with μ_b yields a morphism $Ta \to Tb$, so we get a map $\mu_b \circ T(-) : Ta \times \mathrm{Hom}(a, Tb) \to Tb$.

1.3 quantum systems using monads (without measurement)

Quantum systems with only a finite amount of eigenstates and in the absence of a time-dependent evolution can be described using finite-dimensional complex vector spaces and unitary matrices acting on them. Quantum computing, in large parts, can be implemented in such a system with the unitary matrices describing our basic quantum computational operations, e.g., by using the spin of a particle which only has two eigenstates, up and down.

Given a finite set of eigenstates X, the states of our system live in the free vector space FX generated by X. They are given by the vectors with norm 1, i.e., a state in this system is of the form

$$\sum_{x \in X} \lambda_x |x\rangle$$

with $\lambda_x \in \mathbb{C}$ for all $x \in X$ and $\sum_{x \in X} |\lambda_x|^2 = 1$. We are borrowing the bra-ket-notation from physics and are denoting the vector corresponding to the basis element x as $|x\rangle$. If we were to measure such a state, we would measure the eigenstate $|\tilde{x}\rangle$ with a probability of $|\lambda_{\tilde{x}}|^2$.

We can repackage all of this using the language of monads. Given a set X, we model the free vector space FX as $\mathrm{Map}(X, \mathbb{C})$. The basis vectors $|x\rangle$ are then given by the functions $\delta_x : X \to \mathbb{C}$ which map x to 1 and any other element of X to 0. We have the operations

$$\begin{aligned}
\text{return} :& X \to FX \\
&\text{return } x = |x\rangle \\
\gg\!= :& FX \times \mathrm{Map}(X, FY) \to FY \\
&\left(\sum_{x \in X} \lambda_x |x\rangle \gg\!= f \right) = \sum_{x \in X} \lambda_x f(x)
\end{aligned}$$

which define a monad.

We can define all the usual operations of vector spaces in this framework, such as vector addition, scalar multiplication, the dot product and the tensor product. To give an example, the tensor product is defined using the isomorphism $FX \otimes FY \cong F(X \times Y)$ in the following way

$$\begin{aligned} \otimes : & FX \times FY \to F(X \times Y) \\ & (v \otimes w)(x, y) := v(x) \cdot w(y) \end{aligned}$$

The most important basis for us will be the bit B, i.e., the set containing two elements. We will denote them by 0 and 1 though one can also think of these as the booleans False and True or the states spin up and spin down. A qubit is then the set of states inside the vector space FB, i.e., all complex-linear combinations of $|0\rangle$ and $|1\rangle$ with norm 1. To perform quantum computations on n qubits, we just take the n-fold tensor product of FB with itself which is isomorphic to the vector space $F(B^n)$.

Of particular note is the Einstein-Podolsky-Rosen (EPR) state

$$\frac{1}{\sqrt{2}} (|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle)$$

living inside $F(B^2)$. It cannot be written as the tensor product of two vectors in FB. It is the archetypal example of the concept of entanglement. If we measure the left side of this state and measure $|0\rangle$, we know that the right side has to be $|0\rangle$ as well. If we are to measure $|1\rangle$, we know the other side to be $|1\rangle$. The result of the measurement of the left side immediately affects the state of the right side.

We model the set of linear maps between FX and FY via $\mathrm{Lin}(X, Y) := \mathrm{Map}(X, FY)$. This is a reasonable definition as the free forgetful adjunction between sets and vector spaces gives $\mathrm{Hom}_{\mathbf{Set}}(X, FY) \cong \mathrm{Hom}_{\mathbf{Vect}}(FX, FY)$. We can generate linear maps $\mathrm{Lin}(X, Y)$ in a few different ways. The first one is, given a map $f : X \to Y$ we can compose it with $\mathrm{return} : Y \to FY$ to get an element of $\mathrm{Lin}(X, Y)$. $\mathrm{Not} : B \to B$ maps 0 onto 1 and 1 onto 0. We then also get a linear map Not by composing with $\mathrm{return} : B \to FB$.

Another way of generating linear maps is by defining what they do on the given basis.

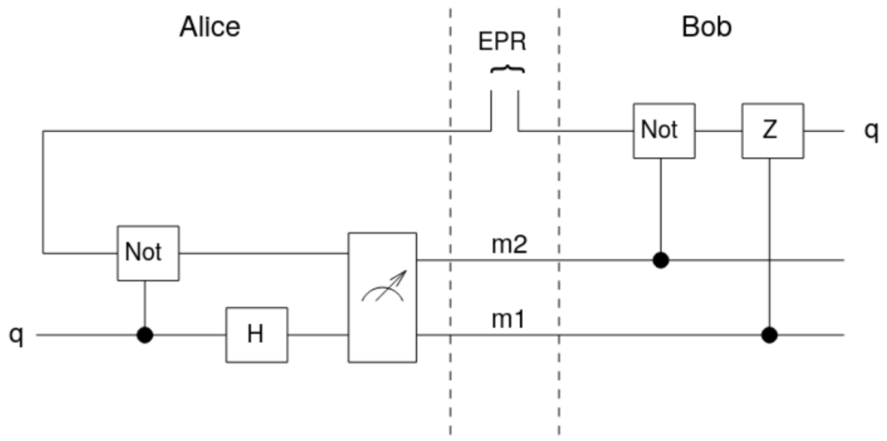
$$\begin{aligned} Z : & B \to FB \in \mathrm{Lin}(B, B) & \qquad H : & B \to FB \in \mathrm{Lin}(B, B) \\ & 0 \mapsto |0\rangle & & 0 \mapsto \frac{1}{\sqrt{2}} (|0\rangle + |1\rangle) \\ & 1 \mapsto -|1\rangle & & 1 \mapsto \frac{1}{\sqrt{2}} (|0\rangle - |1\rangle) \end{aligned}$$

Lastly, given a linear map $f \in \mathrm{Lin}(X, X)$, we can define the linear map controlled $f \in$

$\mathrm{Lin}(B \times X, B \times X)$ via

$$\text{controlled } f\,(b,x) = \begin{cases} |b\rangle \otimes |x\rangle & \text{if } b = 0 \\ |b\rangle \otimes f(x) & \text{if } b = 1 \end{cases}$$

Using these linear maps, we can describe the following quantum algorithm.



This diagram describes an algorithm for transferring the state of a qubit from one person to another. In this setup, Alice has a qubit q and one half of an EPR pair. Bob has the other half of the EPR pair. Alice can transfer the state of q only by transferring classical information.

This diagram is to be read from left to right. The different strands correspond to the different qubits, where the top two strands are in a EPR state and the bottom one is an arbitrary qubit q, so our starting state is

$$\frac{1}{\sqrt{2}}\left(|0\rangle \otimes |0\rangle + |1\rangle \otimes |1\rangle\right) \otimes q.$$

Alice can only interact with the lower strand of the EPR pair, Bob only with the upper strand.

The first that Alice performs is to use a controlled Not gate, where q controls whether Not get applied. She then applies the Hadamard gate H to the lower strands, measures both her qubits and sends the classical information of the measured results $m1$ and $m2$ to Bob. Bob now applies a controlled Not and a controlled Z gate to his half of the EPR pair, controlled by the classical bits $m1$ and $m2$. After doing this, his half of the pair, will now be in the state q that Alice originally had.

We can describe all of the above notions in our monadic setting except for one, we cannot (at least not easily or sensibly) implement measurements monadically.

1.4 density matrices and superoperators

Density matrices are a concept in quantum mechanics that let us represent all quantum states that we have previously discussed. For a quantum system with a finite set of eigenstates X, the density matrices describing states in that system be modeled by the free vector space generated by $X \times X$, so we have $\mathrm{Dens}(X) = F(X \times X)$. We will once again make liberal use of the isomorphism $F(X \times X) \cong FX \otimes FX$.

There is a way of embedding the quantum states we previously discussed into the space of density matrices. These are the so called pure density matrices.

$$\begin{aligned} \mathrm{pureD} : F(X) &\to \mathrm{Dens}(X) \\ \sum_{x \in X} \lambda_x |x\rangle &\mapsto \sum_{x \in X} \sum_{\tilde{x} \in X} \lambda_x \overline{\lambda_{\tilde{x}}}\, |x\rangle \otimes |\tilde{x}\rangle \end{aligned}$$

As the name suggests, we can notate density matrices as matrices.

$$\mathrm{pureD}\ |0\rangle = \begin{pmatrix} 1 & 0 \\ 0 & 0 \end{pmatrix} \quad \mathrm{pureD}\ |1\rangle = \begin{pmatrix} 0 & 0 \\ 0 & 1 \end{pmatrix} \quad \mathrm{pureD}\left(\frac{1}{\sqrt{2}}(|0\rangle + |1\rangle)\right) = \frac{1}{2}\begin{pmatrix} 1 & 1 \\ 1 & 1 \end{pmatrix}$$

Density matrices can also represent some non-pure states that only describe a system probabilistically. The matrix

$$\begin{pmatrix} \frac{1}{2} & 0 \\ 0 & \frac{1}{2} \end{pmatrix} \in \mathrm{Dens}(B)$$

is not in the image of pureD but it nevertheless tells us that our system will be in the state $|0\rangle$ with a probability of $\frac{1}{2}$ and in the state $|1\rangle$ with a probability of $\frac{1}{2}$ when measured.

Superoperators are linear maps between density matrices. We model them as $\mathrm{Super}(X, Y) = \mathrm{Map}(X \times X, \mathrm{Dens}(Y))$. We can embed linear maps into superoperators in the following way.

$$\begin{aligned} \mathrm{lin2super} : \mathrm{Lin}(X, Y) &\to \mathrm{Super}(X, Y) \\ \mathrm{lin2super}\ f\ (x, \tilde{x}) &= f(x) \otimes \overline{f(\tilde{x})} \end{aligned}$$

where $\overline{f(\tilde{x})}$ is defined as taking the complex conjugate of $f(\tilde{x})$ component-wise.

Knowing that we can embed our previously discussed vector spaces and linear maps into the framework of density matrices and superoperators, we can now now do all the calculations from our previous monadic setting in our new setting. Using the fact that density matrices can represent more than just pure states, we can implement the following two functions, which allow us to perform measurements.

$$\mathrm{trL} \in \mathrm{Super}(X \times Y, Y) \qquad \mathrm{meas} \in \mathrm{Super}(X, X \times X)$$

$$\mathrm{trL}((x, y), (\tilde{x}, \tilde{y})) = \begin{cases} |y\rangle \otimes |\tilde{y}\rangle & \text{if } x = \tilde{x} \\ 0 & \text{else} \end{cases} \qquad \mathrm{meas}(x, \tilde{x}) = \begin{cases} |(x, x)\rangle \otimes |(x, x)\rangle & \text{if } x = \tilde{x} \\ 0 & \text{else} \end{cases}$$

The function trL in some sense forgets about the left component of a state. meas produces a pair of vectors corresponding to the collapsed quantum state and the classical measurement. If we first apply meas and then trL to a density matrix produces, we receive a density matrix representing a non-pure state which only records the probability of each possible measurement on its diagonal.

$$\begin{aligned} \mathrm{pureD}\left(\sum_{x\in X}\lambda_x|x\rangle\right) \ggg\!= \mathrm{meas} \ggg\!= \mathrm{trL} &= \sum_{x\in X}|\lambda_x|^2|x\rangle\otimes|x\rangle \\ \mathrm{pureD}\left(\frac{1}{\sqrt{2}}(|0\rangle+|1\rangle)\right) \ggg\!= \mathrm{meas} \ggg\!= \mathrm{trL} &= \begin{pmatrix}\frac{1}{2} & 0\\ 0 & \frac{1}{2}\end{pmatrix} \end{aligned}$$

1.5 superoperators as arrows

While we do believe that superoperators (and therefore measurements) can't be implemented monadically, they can be implemented as arrows which generalize the notion of a monad. Arrows consist of a type constructor Ar which takes two inputs X and Y and returns $\mathrm{Ar}(X,Y)$ together with functions

$$\begin{aligned} \mathrm{arr} &: \mathrm{Map}(X,Y) \to \mathrm{Ar}(X,Y) \\ \ggg &: \mathrm{Ar}(X,Y)\times\mathrm{Ar}(Y,Z) \to \mathrm{Ar}(X,Z) \\ \mathrm{first} &: \mathrm{Ar}(X,Y) \to \mathrm{Ar}(X\times Z, Y\times Z) \end{aligned}$$

which need to satisfy a list of properties. We can now implement superoperators as arrows in the following way.

$$\begin{aligned} \mathrm{arr} &: \mathrm{Map}(X,Y) \to \mathrm{Super}(X,Y) \\ \mathrm{arr}\, f(x,\tilde{x}) &= |f(x)\rangle\otimes|f(\tilde{x})\rangle \\ \ggg &: \mathrm{Super}(X,Y)\times\mathrm{Super}(Y,Z) \to \mathrm{Super}(X,Z) \\ (f \ggg g)(x,\tilde{x}) &= f(x,\tilde{x}) \gg\!= g \\ \mathrm{first} &: \mathrm{Super}(X,Y) \to \mathrm{Super}(X\times Z, Y\times Z) \\ \mathrm{first}\, f((x,z),(\tilde{x},\tilde{z})) &= \pi_{23}\left(f(x,\tilde{x})\otimes|z\rangle\otimes|\tilde{z}\rangle\right) \end{aligned}$$

where $\pi_{23} : FX\otimes FX\otimes FZ\otimes FZ \to FX\otimes FZ\otimes FX\otimes FZ$ just swaps the second and third component in the tensor product.

We now see that the notion of arrows plays nicely with superoperators. Using arr we can transform simple maps into superoperators. $\ggg$ which generalizes the monadic bind operation $\gg\!=$ lets us compose superoperators. Lastly first lets us apply superoperators only on specified parts of a density matrix.

With the framework now established we can even return to the algorithm we previously discussed. Applying pureD to our input we get a density matrix. Using lin2super, we can turn each linear map in the algorithm into a superoperator acting on this density matrix. The measurements Alice performs, can now be modeled by applying meas and trL to the bottom two qubits. At the very end, we can now use trL to discard the bottom two qubits so we're only left with the qubit q which Alice transmitted to Bob.

2 Monads vs. Arrows vs. Relative Monads

2.1 Why can we NOT use monads ?

A short reminder, in functional programming, for a monad m, the multiplication rule corresponds directly to **join :** $m(ma) \to ma$. In reality, it is represented by a mutation **bind :** $ma \to (a \to mb) \to mb$, where $a \to mb$ is a kleisli arrow.

– **Why do we believe monads fail in this setting ?**

One of the main problems within this setting is the typing.

It is shown above that monads work perfectly for the computations within vector spaces, as simply the linear map. However, when it comes to measurements and superoperators, the density map D is desired, it is represented through the bra-ket operation as $D := (a, a)$, which a is the original basis. Categorically, of the form (A, A^{op}).

Hence unlike linear maps:

$$a \to mb$$

Superoperators are:

$$(a, a) \to mb$$

They do NOT form Kliesli arrows. Because given two kleisli arrows $f : A \to MB$, $g : B \to MC$, to form the kleisli category, we need the codomain of f to match the domain of g. Not only it is lack of well defined identity, the composition also doesn't work out: Suppose we have $f : A \times A \to MB$ and $g : B \times B \to MC$, g is expecting $B \times B$ as input.

To attack this problem, there are several ways we could try out:

- When M is a monoidal monad, and it interacts nicely with the monoidal product. which requires too much additional structures.
- Using Comonoids:

  Every object A is equipped with a comonoid structure (i.e., a diagonal map $A \to A \times A$ and a counit $A \to 1$. (Any Copy-Discard Category) would work. But again it's far from what we are desiring.
- Use some other structures such as arrows.

The paper gives a solution using arrows, observing that the arrows, which intake inputs of the form of a pair (A, A^{op}), are also generalization of strong monads (*Caution : it does not capture non-strong monads*). Typically, the monads in functional programming languages are necessarily strong. It solve the typing problem elegantly and also naturally extends what strong monads could do simply by using the kleisli arrows.

A more general way of seeing this mismatch is simply that, measurements do not live in vector spaces, which monads could model. Instead, they live in (for finite dimensional quantum theory), finite dimensional vector spaces. And finite (or even infinite) dimensional vector spaces, instead of being an endofuctor, is a (forgetful) functor :

$$U : \mathbf{FinVect} \to \mathbf{Set}$$

It is not an endofuctor hence could not directly form a monad.

However, this statement received some doubts from the quantum computing community, especially, Dr. Vladimir Zamdzhiev from QuaCS team questioned that:

– Is it really true that one could say monads do not work for functional quantum computing ?

Because clearly there has been several breakthroughs within the community, that people successfully constructed ways to model non-classic computations like superoperators, and found adjunction situation between functors.

Especially, Peter Selinger and Benoit Valiron's quantum lambda calculus (Benoit Valiron's PhD thesis) potentially gave the model including the (co)monad along with the calculus, however they did not find it in quantum setting back then. Further Kenta Cho and Abraham Westerbaan's work `https://arxiv.org/pdf/1603.02133`, found a model based on von Neumann algebras, shows that there is indeed an adjunction between $\mathbf{vNA}^{OP}_{MIU}$ and $\mathbf{Set}$. More explicitly, between the forgetful functor:

$$F : \mathbf{vNA}^{OP}_{MIU} \xrightarrow{nsp} \mathbf{Set}$$

and free vector space functor:

$$U : \mathbf{Set} \xrightarrow{l^{\infty}} \mathbf{vNA}^{OP}_{MIU}$$

F is left adjoint to U, hence we could construct a monad

$$T = l^{\infty} \circ nsp$$

from this adjunction.

Some remarks about this state monad is, that it controls the process, instead of being a computational monad. And some interesting comments from Peter Selinger via Benoit Valiron : *My supervisor used to say that using vNA is cheating: it is big enough to contain "everything and the kitchen sink".*

But as one can easily observe, if we restrict into finite dimension, U could not be right adjoint to F, because U has finite dimensions, could not land in F which has infinite dimensions.

– Conclusion: Monads do not exist in finite dimension setting, but do exist in infinite dimension setting, arise from the adjunction.

2.2 A step further from arrows : Relative Monads

But real life quantum computing are normally finite dimensional, when quantum measurements yield discrete outcomes, while infinite dimensions would also imply continuous observables (e.g., position/momentum), which are harder to control.

If we view the problem of the failure of using monads as the incapability of constructing a monad from a functor mapping two different categories which, one lives in finite dimension and other infinite dimension, then as the title of the paper says, monads need not be endofunctors `https://arxiv.org/abs/1412.7148` ! The **Relative Monads** it defines yield another solution to the problem which has more monadic tastes.

A Relative Monad is defined for a functor J between two categories $\mathbf{J}$ and $\mathbf{C}$, data and laws of a relative monad are exactly as those of a monad, except the occurrences of the first $\mathbf{C}$ is replaced as $\mathbf{J}$.

We can even generalize arrows to relative monads, technically, an arrow on $\mathbf{J}$ is the same thing as a relative monad on the Yoneda embedding $Y \in \mathbf{J} \to \mathbf{J}^{op}, \mathbf{Set}]$. Compare to arrows, relative monads have all the nice monadic components such as kleisli and Eilenberg-Moore constructions. It works perfectly with finite dimensional vector spaces.

For example, consider the finite-demensional vector spaces, see Example 1.1 in `https://arxiv.org/abs/1412.7148`. Using this, we would be able to structure the quantum computations further with the relative monad $(\mathbf{Vec}, \eta, (-)^*)$, the explicit construction will be the future work for our group.

Moreover, even in the infinite dimension setting, compared to the monad arises from the adjunction, we believe relative monads also exist and it potentially provides alternatively a more trivial and easier-to-manipulate structure, which is more natural in functional programming to handle computational effects.