Intuition-enhancing GUI for visual programming

Vasile Adrian Rosian
Department of Computer Science
"Babes-Bolyai" University Cluj-Napoca
adrian.rosian@gmail.com

Abstract

While searching for a mechanism for combining free constructs and cofree interpreters with monad-comonad adjunctions for GUIs ((Freeman, 2017) and (Xavier, Da, Bigonha, & Freeman, 2018)) to permit transitioning the state cursor of a GUI application and constructing a DSL using free monads for this state cursor transition that is later on given Haskell semantics through string diagrams (Coecke & Kissinger, 2018) we arrived at the problem of enhancing usability of the solution beyond the advantages given by composability as a means of reducing complexity.

The semantics of string diagrams are limited in the degrees of freedom available to them to enhance the "intuition" of the user because of their 2D structure and lack of semantics for positioning. We explore here similar semantics in a 3D space using the idea that data types are "circles" positioned at some sort of "distance" from a conscious observer that gravitate towards a "center" given by an "input-output" tensor, much like magnetic fields around a coil. The combination gives birth to a torus shape where programs are closed transversal loops that merge into "vortexes" (or "spirals") on the surface of the torus. Enhancement the intuition is then available when giving semantics to size, rotation speeds, traversal speed, color, sound - in a consistent manner - in a VR environment.

1. GUIs with monads and comonads

1.1. State, store, pairing

In category theory an adjunction is a relaxation of the definition of an equivalence of categories. Seen from the perspective of hom-sets an adjunction is a bijection of those sets. Formally:

$$
\begin{aligned}
&F : \mathscr{C} \to \mathscr{D};\ \mathscr{C}, \mathscr{D} \in Cat \\
&U : \mathscr{D} \to \mathscr{C} \\
&(\eta, \varepsilon) : F \vdash U \\
&\varepsilon : F \circ U \to 1_{\mathscr{D}} \\
&\eta : 1_{\mathscr{C}} \to U \circ F \\
&or \\
&hom_{\mathscr{C}}(X, UY) \cong hom_{\mathscr{D}}(FX, Y)
\end{aligned}
$$

The η and ε natural transformations between the free (F) and forgetful (U) functors are the familiar monadic unit and co-monadic co-unit.

Intuitively, an adjunction specifies to the maximum extent possible that two categories are quasi-identical in structure (with determined transforms). Freeman makes explicit such a comparison between state and the store of interfaces of a program.

As such, transitioning state in a state monad is akin to switching screens in the space of all screens of an application, effectively representing reactive applications through a pairing (Kmett, 2011) of functors that "cancel" each other's contexts.

Xavier et al. makes use of the pairing under the shape of Kmett's *zapWithAdjunction* morphism to express the connection between switching the screens of a comonadic interface and the transitioning of application state through monadic actions.

1.2. Combining GUIs

Xavier et al. gives two examples fo combining GUIs expressed as comonads: a binary-driven sum type for interfaces that are switched in place and the Freeman-inspired Day convolution of two functors to express adjoint UIs. For hierarchical components, however, the solution given by Xavier et al. is still comonad transformers.

A hierarchy of components (like a modal inside a parent window) is given by:

```
data StoreT s w a = StoreT
                      (w (s -> a)) s
instance Comonad w
  => Comonad (StoreT s w) where
      extract (StoreT wtrans s)
       = extract wtrans s
      duplicate (StoreT wtrans s) =
        StoreT (extend StoreT wtrans) s
```

A side-by-side joining of two components is represented by a *Day* convolution (Kmett, 2016) :

```
data Day f g a = forall b c.
  Day (f b) (g c) (b -> c -> a)
```

and a pair of interfaces where one replaces the other based on a binary toggle is given by (Xavier et al., 2018):

```
data Sum f g a = Sum Bool (f a) (g a)
```

where the datatype *a* is the type of the value written to the component state once the actions in the interface have been executed.

To be noted that language limitations in Haskell do not allow the compiler to discriminate on types and thus provide a correct interpretation in the case where DSL branches are not complete on the DSL continuation type, which is often the case for complete programming languages (yao Xia, 2019).

A good approach based on the GADT technique is implemented based on Wu and Schrijvers and is available in the Polysemy library `https://github.com/polysemy-research/polysemy`.

1.3. State transition function boundaries

With these primitive combinators one can build a comonadic tree that will pair with a *Co* monad obtained using Kmett:

```
co :: Functor w => (forall r.
  w (a -> r) -> r) -> Co w a
runCo :: Functor w =>
  Co w a -> w (a -> r) -> r
```

If *a* and *r* are state values then the transition function is reduced to a function of modifying state while optionally running some effects.

2. Representing state transforms

String diagrams (Coecke & Kissinger, 2018) are a way of graphically representing strict monoidal categories that ensures that the graphical operations on the wires and boxes representing typed values and morphisms make sense in the underlying category.

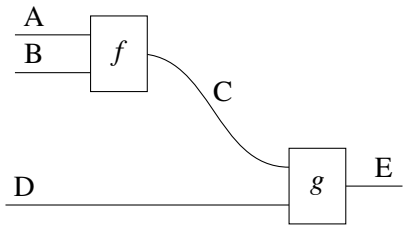


Figure 1 – Wires are typed values, boxes are morphisms

2.1. Standard transforms

In string diagrams values are typed wires and boxes are morphisms.

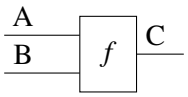


Figure 2 – Morphism $f : A \otimes B \to C$

Identities are represented by "ghost" boxes on the wires, effectively "not transforming" the value on the wire. In (2) any of the wires might as well be transformed by an identity.

Composition (3) is represented by connecting outputs to inputs and parallel composition represents the monoidal product $\otimes$.

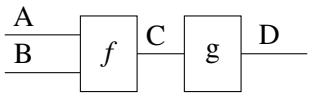


Figure 3 – $g \circ f : A \otimes B \to D$

2.2. Compositions

As mentioned, simple composition can be performed like in (3) but the output of a box can be connected to part of the input of another box like in (1).

However, we can also split the monoidal product of a box output to provide partial inputs to subsequent functions, like in (4).

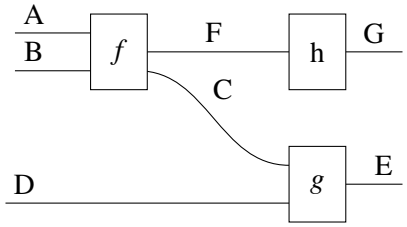
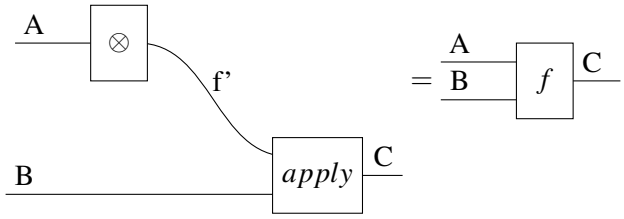


Figure 4 – Monoidal product splitting

Composition helps prove the expressiveness of string diagrams by showing easy equivalences:

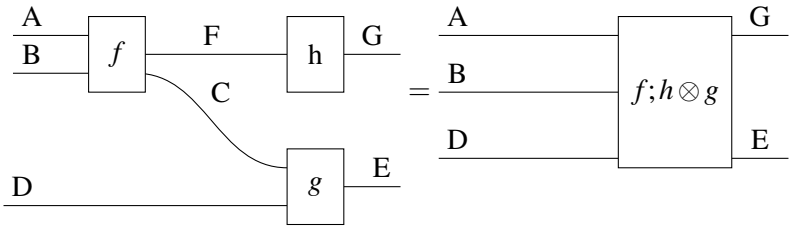
2.3. Transform resolution

The graphical language allows collapsing regions of the graph into boxes by mapping input morphisms of the region into inputs to the collapsed box and outputs to box outputs, effectively allowing to "zoom" out of a graphical view while retaining typing information. The approach can be found in the graphical language Enso Luna (Team, 2020).



(1)

Figure 5 – Morphism $f : A \otimes B \to C$ expressed as curry-ing



(2)

Figure 6 – It is possible to "zoom out" of a diagram

3. Representing effects

Effects in pure functional programming languages such as Haskell are achieved via monads (Moggi, 1991). However, Moggi's definition is not helpful in identifying the right string diagram candidate for the monad representation.

It is Launchbury and Peyton Jones' implementation of the IO monad for Haskell using state threads that points us in the right direction. In the transformer diagram the state thread transformer is represented as taking in the "world" state in addition to normal inputs and returning the modified world alongside regular outputs.

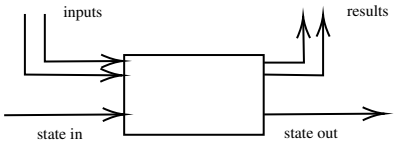


Figure 7 – The ST transformer used as a basis for IO in Haskell

This is consistent with the categorical view of a monad when an adjunction is involved. The ε and η natural transformations are morphisms in their respective categories that get their semantics from the functorial relation with the exterior of the category. To be noted that also in this representation we are talking about functors.

The round trip provided by an adjunction provides a nice way of interpreting two functorial relations (the "right" and the "what is left" directions) as one relation in the original category through the means of a natural transformation (a monad is, after all, a monoid for endofunctors - with "endo" being key here). In other words, if category $\mathscr{D}$ is planet Earth and there is some distant, M-class planet discovered called $\mathscr{C}$, we would be able to describe the lifestyle of the inhabitants of $\mathscr{C}$ using regular actions from the Earth only thanks to the adjunction between the two planets. Regular actions on Earth get new meanings based on the discovery of $\mathscr{C}$ and analogies we can make.

When we talk about some communication (transmission) between Earth and $\mathscr{C}$ we are only interested of either the failure of the transmission or the possible effects on Earth of a successful communication (say we instructed the inhabitants of $\mathscr{C}$ to setup a video feed - we are only interested if we managed to get a new TV channel where we can see the strange new world of $\mathscr{C}$). Which is to say that an "effect" is only interesting in terms of the results it produces in the originating process.

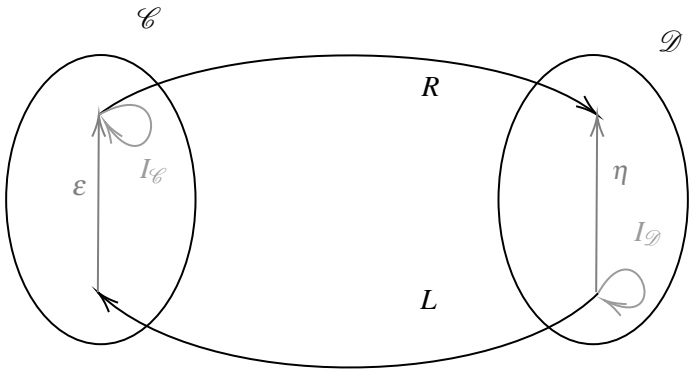


Figure 8 – Monads from adjunctions in category theory

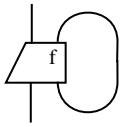


Figure 9 – A monadic effect candidate is the partial trace of a process

As such, the right candidate for monad representation and thus handling of effects in the string diagram language is a partial trace of a process, represented by the process box with the regular part of inputs/outputs and the traced "world" loop that synchronizes the effect on the world with the linked state obtained.

4. Intuition-enhancing representation

4.1. The torus

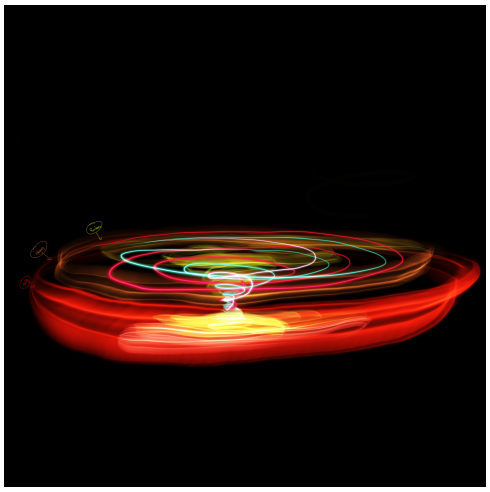


Figure 10 – The program torus attempt at artistic rendering

The semantics of string diagrams are limited in the degrees of freedom available to them to enhance the "intuition" of the user because of their 2D structure and lack of semantics for positioning. We propose similar semantics in a 3D space using the idea that data types are "circles" positioned at some sort of "distance" from a conscious observer that gravitate towards a "center" given by an "input-output" tensor, much like magnetic fields around a coil. The combination gives birth to a torus shape where programs are closed transversal loops that merge into "vortexes" (or "spirals") on the surface of the torus.

Enhancement the intuition is then available when giving semantics to size, rotation speeds, traversal speed, color, sound - in a consistent manner - in a VR environment.

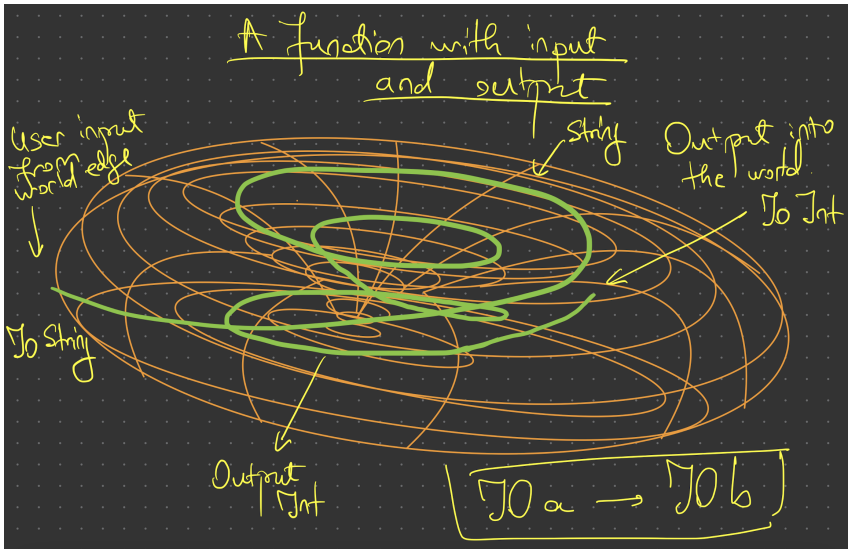


Figure 11 – Function with IO

The semantics of the string diagrams are conserved across the representation, and, even better, asynchronous executions of programs, in terms of crossing the "World edge" are better visualized.

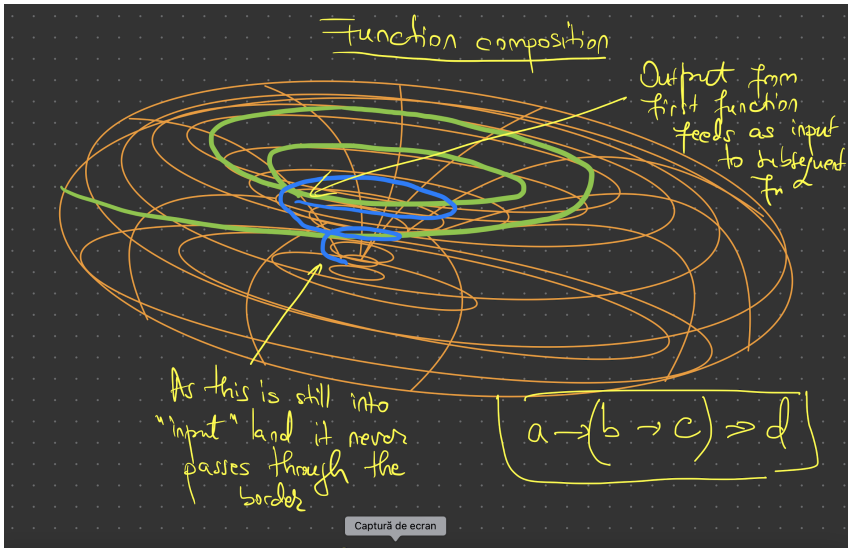


Figure 12 – Function composition

4.2. Visuals

Similar to 3D game worlds, vibration is the key concept used here for conveying and enhancing intuition. When objects in the world vibrate in the same color spectrum they are intuitively semantically close.

As such, the torus is color-coded from exterior "World edge" to interior vortex center to represent basic data types. The colors are conveyed using particle effects (diffuse luminescent smoke) and are presented as "bands" where function "loops" can take place.

Functions are light lines following the surface of the torus. If they start at the world edge they will start as dark red light, otherwise they start as the color of the starting torus band. The light within pulsates at the rate of the starting color. Function lines loop in the band of the data-type where the input/output is. Function lines can join other lines or can split. Joining and splitting modifies both pulse frequency and

color blending.

Pulses of light give the overall impression of spiral rotations which, together with golden-ratio spiral sizes and steps should create a hypnotic effect on the user, stimulating the building instinct.

During construction of the program elements of the construction will either harmonize pulsation and color or will cause electric-like (or light saber-like) sparkles if they do not match in position.

All operations of the VR environment are available, such as resizes, rotations, zoom-in and out, navigation.

4.3. Sound

When it comes to sound there is an intrinsic vibration of the space surrounding the torus that will give the impression, especially around torus world edges, of "generated" light. It is ethereal and omnipresent, keeping the attention and focus on the light torus in front of the user.

Type bands on the torus will also match sound harmonics according to some (hopefully) coherent mapping. Approaching the type band would intensify the sound while distancing oneself from it would reduce the volume of the band.

Function spirals would have matching sounds, but in a higher range and with an added electric buzzing effect. Function dis-harmonies on joins would cause the corresponding electric crackle of a short-circuit.

Sound would be spatial, causing pass-by Doppler effects and would be localized as source for each element of the program.

4.4. Haptics

Normal VR haptics would indicate matching of functions at specific locations (or lack of thereof). Depending on zoom level and position of navigation in the program space, gyroscopic resistance can be factored in for moving spirals around the program space.

5. Future work

The presented work is theoretical and very early-stage, leaving plenty of room for development.

Although intuitively the semantics of the work match string diagrams, equivalence still needs to be demonstrated, especially around partial traces and equivalent matrix transforms. Completeness of the representation is also a field of concern, especially around the four horsemen of IO, Exceptions, Continuations and Randomness. Special attention needs to be paid to bounding mechanics, as outlined by the work of Statebox team (The Statebox Team, 2018).

In the interface part, GUI considerations for the user interface, such as library of components access, layout building mechanics, Day convolution function supply and improvements to intuitive clues still can benefit from a lot of attention.

6. Acknowledgements

A sizable thank you to my coordinating professor, dr. Horia F. Pop for constructive feedback and patience. A special thanks also goes to Radu Ometita for encouraging me to better understand comonads, Claudiu Ceia and George Cosma for taking the time to hear and criticize my musings as well as to Iuliana Rosian for her continuous support.

7. References

Coecke, B., & Kissinger, A. (2018). Picturing quantum processes: A first course on quantum theory and diagrammatic reasoning. In *Lecture notes in computer science (including subseries lecture notes in artificial intelligence and lecture notes in bioinformatics).* doi: 10.1007/978-3-319-91376-6_6

Freeman, P. (2017). *Declarative UIs are the Future — And the Future is Comonadic!* Retrieved from `https://functorial.com/the-future-is-comonadic/main.pdf`

Kmett, E. (2011). *Monads from comonads.* Retrieved from `http://comonad.com/reader/`

2011/monads-from-comonads/

Kmett, E. (2016). *Kan extensions package.* Retrieved from https://hackage.haskell.org/package/kan-extensions-5.0.1

Launchbury, J., & Peyton Jones, S. L. (1994, June). Lazy functional state threads. *SIGPLAN Not.*, *29*(6), 24–35. Retrieved from https://doi.org/10.1145/773473.178246 doi: 10.1145/773473.178246

Moggi, E. (1991, July). Notions of computation and monads. *Inf. Comput.*, *93*(1), 55–92. Retrieved from https://doi.org/10.1016/0890-5401(91)90052-4 doi: 10.1016/0890-5401(91)90052-4

Team, T. E. (2020). *Enso: The Syntax.* https://github.com/luna/enso/blob/master/doc/syntax/specification/syntax.md. ([Online; accessed 15-May-2020])

The Statebox Team. (2018). *The Mathematical Specification of the Statebox Language* (Tech. Rep.). Retrieved from https://statebox.io

Wu, N., & Schrijvers, T. (2015). Fusion for free efficient algebraic effect handlers. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *9129*, 302–322. doi: 10.1007/978-3-319-19797-5_15

Xavier, A., Da, R., Bigonha, S., & Freeman, P. (2018). *A Real-World Application with a Comonadic User Interface.* Retrieved from https://arthurxavierx.github.io/RealWorldAppComonadicUI.pdf

yao Xia, L. (2019). *Free monads of free monads.* https://blog.poisson.chat/posts/2019-06-09-free-monads-free-monads.html. ([Online; accessed 14-May-2020])