Monadic Design for Universal Systems

Nick Rossiter
Visiting Fellow
Computing Science and Digital Technologies
Northumbria University

ANPA 37 (August 2016)

Outline of Presentation

- Basic categorical facilities identified for the Universe
  - The Topos (structural data-type)
  - The Monad (process)
- Applying the monad to a topos
  - Alternative Techniques
  - Application
- Discussion

The Topos – Structural Data-type

- Based on Cartesian Closed Category (CCC)
  - Products; Closure at top; Connectivity (exponentials); Internal Logic; Identity; Interchangeability of levels
- If we add:
  - Subobject classifier
  - Internal logic of Heyting (intuitionistic)
  - Reflective subtopos (query closure)
- We get a Topos

Topos: further work identified 2-years ago and now

- Data Process
  - Queries within a topos – use of subobject classifier, particularly with power objects
  - Examples of Heyting intuitionistic logic
  - Applying process to a topos
- Database design
  - Co-cartesian approach
  - Pasting of pullbacks
  - Recursive pullbacks
  - Allegories

Progress

Data Process	
Queries within a topos	Subobject classifier extended to power-objects for generality
Heyting examples, internal logic	Stalled for while but now restarted
External process on a topos	Today's topic, monads
Database Design	
Co-cartesian	For normalisation, not now a priority as thought to be categorification below 5NF
Pasting of pullbacks	Expressed in complex, more realistic design, satisfies 5NF
Recursive pullbacks	In progress
Allegories	Explored, not useful in natural IS but significant for interoperability

Pullback - Single Relationship

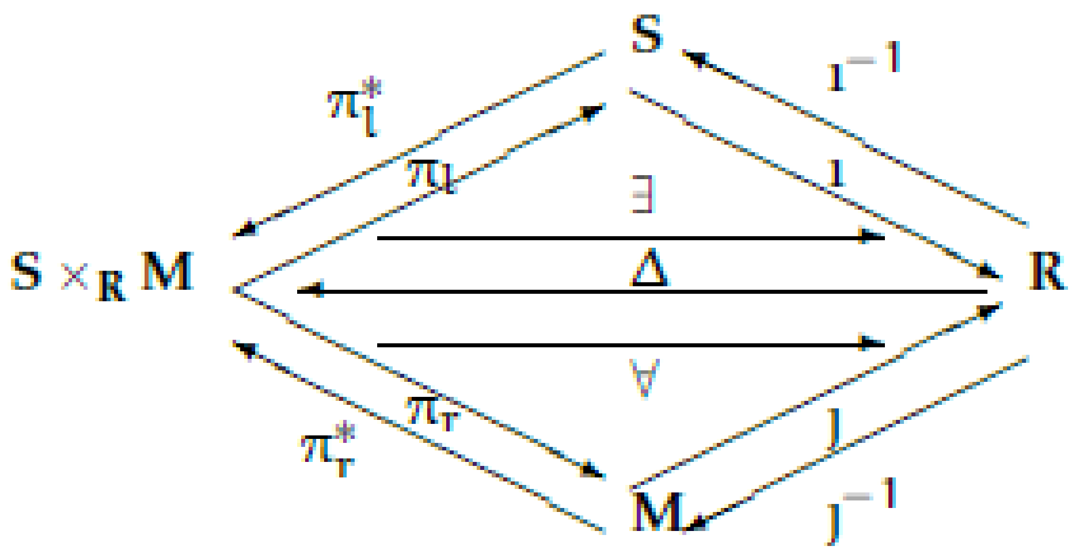


Figure 8: Single Pullback: $\mathbf{S} \times_{\mathbf{R}} \mathbf{M}$; **S** student, **M** module, **R** result

Pullback – x6 Multiple Relationship

$$
\begin{tikzcd}
(((S \times_R M) \times_M D) \times_D U) \arrow[r, "\pi_r"] \arrow[d, "\pi_l"'] \arrow[dr, phantom, "P3"] & U \arrow[d] \\
((S \times_R M) \times_M D) \arrow[r, "\pi_r'"] \arrow[d, "\pi_l'"'] \arrow[dr, phantom, "P2"] & D \arrow[d] \\
(S \times_R M) \arrow[r, "\pi_r''"] \arrow[d, "\pi_l''"'] \arrow[dr, phantom, "P1"] & M \arrow[d] \\
S \arrow[r] & R
\end{tikzcd}
$$

A pasted pullback
Is only a valid pullback
If all inner and outer
diagrams are pullbacks

Figure 9: Pasted Pullback Category: $(((\mathbf{S} \times_{\mathbf{R}} \mathbf{M}) \times_{\mathbf{M}} \mathbf{D}) \times_{\mathbf{D}} \mathbf{U})$; **U** university, **D** department, **S** student, **M** module, **R** result; $P1, P2, P3$ are labels for diagrams for purpose of discussion

Pullback – x6 with Logic

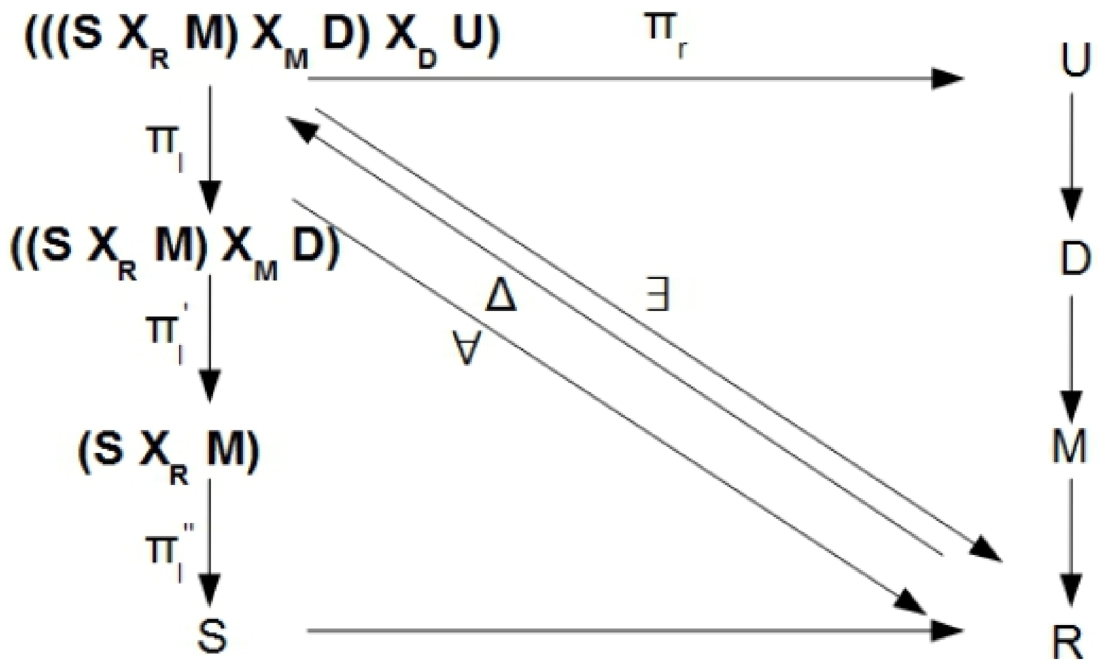


Figure 10: Pasted Pullback Category $(((\mathbf{S} \times_{\mathbf{R}} \mathbf{M}) \times_{\mathbf{M}} \mathbf{D}) \times_{\mathbf{D}} \mathbf{U})$; **U** university, **D** department, **S** student, **M** module, **R** result; intuitionistic logic $\exists \dashv \Delta \dashv \forall$ for the outer pullback diagram

Pullback – x7 Not Natural

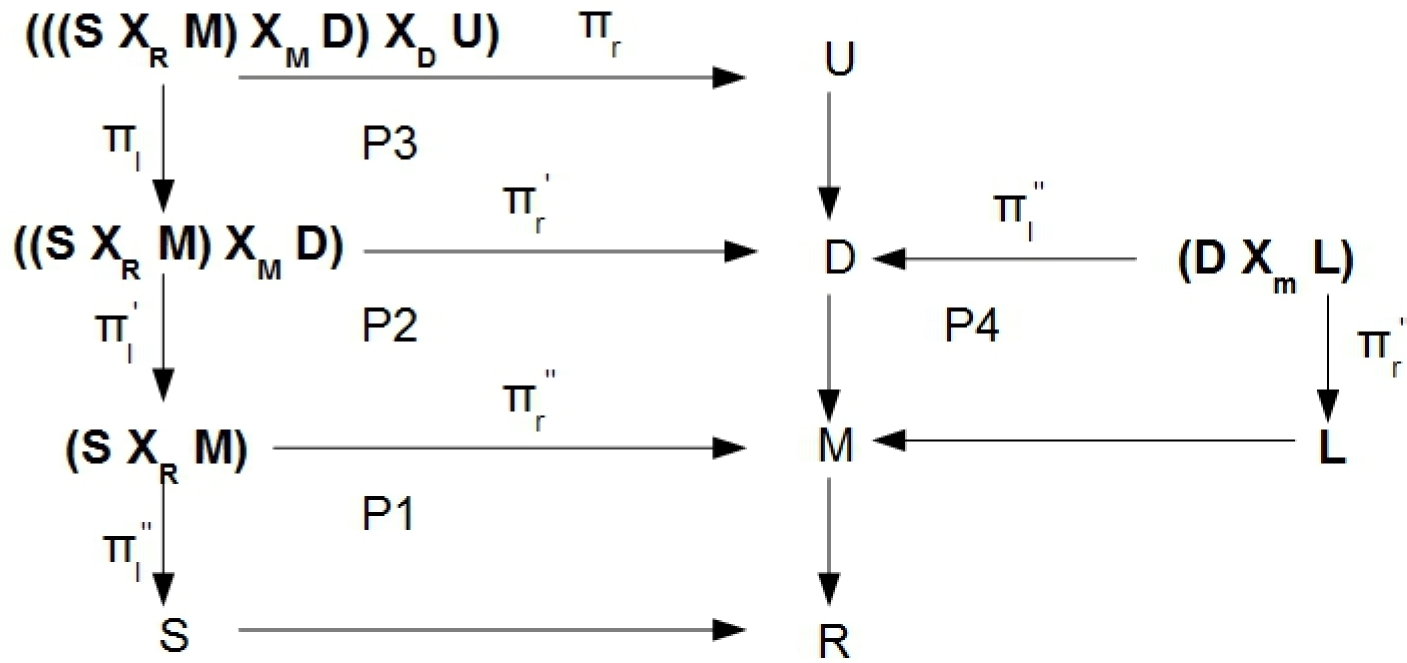


Figure 11: Pasted Construction $(((\mathbf{S} \times_{\mathbf{R}} \mathbf{M}) \times_{\mathbf{M}} \mathbf{D}) \times_{\mathbf{D}} \mathbf{U})$; **U** university, **D** department, **L** lecturer, **S** student, **M** module, **R** result; P1, P2, P3, P4 are labels for diagrams for purpose of discussion; not a valid pullback category as not natural

Pullback – x10 Natural

$$
\begin{tikzcd}
(((S \; X_R \; M) \; X_M \; (D \; X \; L)) \; X_{D \, X \, L} \; D) \; X_D \; U \arrow[r, "\pi_r"] \arrow[d, "\pi_l"'] \arrow[dr, phantom, "P4"] & U \arrow[d] \\
((S \; X_R \; M) \; X_M \; (D \; X \; L)) \; X_{D \, X \, L} \; D \arrow[r, "\pi_r'"] \arrow[d, "\pi_l'"'] \arrow[dr, phantom, "P3"] & D \arrow[d] \\
(S \; X_R \; M) \; X_M \; (D \; X \; L) \arrow[r, "\pi_r''"] \arrow[d, "\pi_l''"'] \arrow[dr, phantom, "P2"] & D \; X \; L \arrow[d] \\
(S \; X_R \; M) \arrow[r, "\pi_r'''"] \arrow[d, "\pi_l'''"'] \arrow[dr, phantom, "P1"] & M \arrow[d] \\
S \arrow[r] & R
\end{tikzcd}
$$

Figure 12: Pasted Pullback Category $(((\mathbf{S} \times_{\mathbf{R}} \mathbf{M}) \times_{\mathbf{M}} (\mathbf{D} \times \mathbf{L})) \times_{\mathbf{D}\times\mathbf{L}} \mathbf{D})\mathbf{X}_{\mathbf{D}}\mathbf{U}$; **U** university, **D** department, **L** lecturer, **S** student, **M** module, **R** result; P1, P2, P3, P4 are labels for diagrams for purpose of discussion

Subobject Classifier – Searching within a Topos - Boolean example

$$
\begin{tikzcd}
U \arrow[r] \arrow[d, "j"'] & 1_{topos} \arrow[d, "true"] \\
S \arrow[r, "X_j"'] & \Omega\ \{0,1\}
\end{tikzcd}
$$

Simple database query in category theory style

$\Omega\ \{0,1\}$ is subobject classifier; subobjects classified as either 0 or 1

X_j characteristic function is query mapping from object S to {0,1), false or true

1_{topos} is terminal object of topos (handle on topos)

j is mapping from subtopos U (result of query) to object S

Diagram is actually a pullback of *true* along $X_{j.}$

U is $1_{topos}\ X_{\Omega\{0,1\}}\ S$

U is the identity of the subtopos, giving query closure

External Process

- Metaphysics (Whitehead)
- Transaction (universe, information system)
- Activity
  - Can be very complex but the whole is viewed as atomic – binary outcome – succeed or fail
  - Before and after states must be consistent in terms of rules
  - Intermediate results are not revealed to others
  - Results persist after end

Transaction is standard way of defining a Process

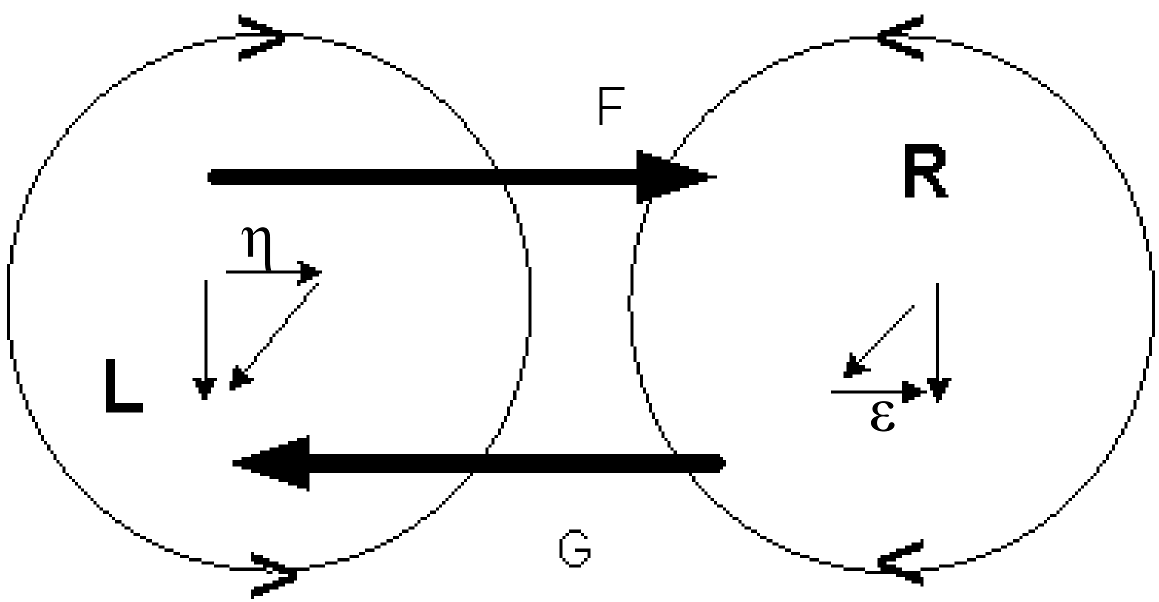
- Principles of ACID
  - Atomicity, Consistency, Isolation, Durability
- Logical technique for controlling the physical world
  - e.g. banking transaction
- Requires three cycles of adjointness between initial and target state
- First two for atomicity, consistency and isolation
  - First makes changes; second reviews changes
- Third for durability

Transaction in Category Theory

- In earlier work (ANPA 2010) we used adjointness to represent a transaction
  - Employing multiple cycles to capture ACID
- The aim now is to abstract this work using the monad, which we earlier described as the way forward
- The monad is an extension of the monoid to multiple levels
  - Monoid: $M \times M \rightarrow M$, $1 \rightarrow M$ (binary multiplication, unit)

Multiple 'Cycles' to represent adjointness

- Three ‘cycles’ GFGFGF:
  - Assessing unit η in L and counit ε in R to ensure overall consistency
  - 'Cycles' are performed simultaneously (a snap, not each cycle in turn)



$F \dashv G$

$\eta: 1_L \rightarrow GF(L)$ $\varepsilon: FG(R) \rightarrow 1_R$

Promising Technique - Monad

- The monad is used in pure mathematics for representing process
  - Has 3 'cycles' of iteration to give consistency
- The monad is also used in functional programming to formulate the process in an abstract data-type
  - In the Haskell language the monad is a first-class construction
    - Haskell B. Curry transformed functions through currying in the λ-calculus
    - The Blockchain transaction system for Bitcoin and more recently other finance houses uses monads via Haskell
      - Reason quoted is it's a simple and clean technique

Monad/Comonad Overview

- Functionality:
  - Monad
    - $T^3 \rightarrow T^2 \rightarrow T$ (multiplication)
    - 3 'cycles' of T, looking back
  - Comonad (dual of monad)
    - $S \rightarrow S^2 \rightarrow S^3$ (comultiplication)
    - 3 'cycles' of S, looking forward
- Objects:
  - An endofunctor on a category X

Using the Monad Approach

- A monad is a 4-cell <1,2,3,4>
  - 1 is a category X
  - 2 is an endofunctor (T: X $\rightarrow$ X, functor with same source and target)
  - 3 is the unit of adjunction η: $1_X \rightarrow T$ (change, looking forward)
  - 4 is the multiplication μ: T X T $\rightarrow$ T (change, looking back)
- A monad is therefore <X, T, η, μ>

The Monad as a 'triple'

- A monad is sometimes called a triple as by Barr & Wells. Term disliked by some as too set theoretic, e.g. Mac Lane
- Why a triple when 4 terms above?
- <X, T, η, μ> is reduced to
  - < T, η, μ> as category X is implicit in T
- True to the spirit of category theory a monad works over 3 levels, as 3 levels gives naturality
- So the laws we are going to see involve T (endofunctor performed once), T^2 (performed twice) and T^3 (performed 3 times)
- Note this multiple performance matches our transaction approach, outlined earlier with GF perfomed 3 times

The Comonad

- The dual of the monad
- A comonad is a 4-cell <1,2,3,4>
  - 1 is a category X
  - 2 is an endofunctor ($S: X \to X$, functor with same source and target, S is dual of T)
  - 3 is the counit of adjunction $\varepsilon: S \to 1_X$ (change, looking back)
  - 4 is the comultiplication $\delta: S \to S\ X\ S$ (change, looking forward)
- A comonad is therefore $<X, S, \varepsilon, \delta>$ or $<S, \varepsilon, \delta>$

Laws for the Monad

- Book-keeping
- Associative Law
- Unit Law

Associative Law for Monad

- The laws involve T^3 (3 'cycles') with the Associative law:

$$
\begin{tikzcd}
T^3 \arrow[r, "T\mu"] \arrow[d, "\mu T"'] & T^2 \arrow[d, "\mu"] \\
T^2 \arrow[r, "\mu"'] & T
\end{tikzcd}
$$

(a)

$$
\begin{tikzcd}
S \arrow[r, "\delta"] \arrow[d, "\delta"'] & S^2 \arrow[d, "S\delta"] \\
S^2 \arrow[r, "\delta S"'] & S^3
\end{tikzcd}
$$

(b)

a) Associative law for monad <T,η,μ >; b) Associative law for comonad <S,ε,δ>

Unitary Law for Monads

- The diagram commutes

$$\begin{tikzcd}
IT \arrow[r, "\eta T"] \arrow[d, "="'] & T^2 \arrow[d, "\mu"] & TI \arrow[l, "T\eta"'] \arrow[d, "="] \\
T \arrow[r, "="'] & T & T \arrow[l, "="]
\end{tikzcd}$$

Figure 13: Left and Right Unitary Laws for Monad $T =< T, \eta, \mu >$

Monad can be based on an adjunction

- The transaction involves GF, a pair of adjoint functors F -| G
  - F: X → Y
  - G: Y → X
- GF is an endofunctor as category X is both source and target
- So T is GF (for monad)
- And S is FG (for comonad)

3-cell descriptors with adjoints

- The 3-cell monad < T, η, μ>
  - is written <GF, η, GεF> (last up a level for multiplication)
- The 3-cell comonad <S, ε, δ>
  - is written <FG, ε, FηG> (last up a level for comultiplication)

Terminology

- A monad is often simply addressed by its endofunctor.
  - So $<T, \eta, \mu>$ is called the monad T
- Similarly for the comonad
  - $<S, \varepsilon, \delta>$ is called the comonad S
- It's a synecdoche

Operating on a Topos

- The operation is simple:
  - $T: E \rightarrow E$
    - where T is the monad $\langle GF, \eta, G\varepsilon F \rangle$ in E, the topos, with input and output types the same
- The extension (data values) will vary but the intension (definition of type) remains the same
- Closure is achieved as the type is preserved

The T-algebras – Changing the Definition

- More fundamental change to the operand (X or E)
- Produces a new consistent state of adjunction with modified intension
- The T-algebras manipulate the category X, when defined within a monad T
- They were developed in work by Eilenberg & Moore published in 1965.

T-algebra defined

- For a category X, not necessarily a topos, in the monad $\langle X, T, \eta, \mu \rangle$, the effect is to obtain:
  - $\langle G^T F^T, \eta^T, G^T \varepsilon^T F^T \rangle : X \rightarrow X^T$
- That is a new monad adjunction $F^T \dashv G^T$ is defined to accommodate the changed category X^T
- For a topos E, this is equivalent to a change to E^T
  - $\langle G^T F^T, \eta^T, G^T \varepsilon^T F^T \rangle : E \rightarrow E^T$

The T-algebra

- For a monad <T, η, μ> in X
- A T-algebra is:
  - <x, h>
- Where x is an an object in X
- And h: Tx → x is the structure map of the algebra
- Such that the following diagrams commute.

T-algebra: Associative/Unitary Laws

Associative

$$\begin{tikzcd} T^2x \arrow[r, "Th"] \arrow[d, "\mu_x"'] & Tx \arrow[d, "h"] \\ Tx \arrow[r, "h"'] & x \end{tikzcd}$$

Unitary

$$\begin{tikzcd} x \arrow[r, "\eta_x"] \arrow[dr, "1"'] & Tx \arrow[d, "h"] \\ & x \end{tikzcd}$$

Both diagrams must commute for T to be a monad

Other Monadic features

- Kleisli Category of a Monad
  - Transforms a monad into a form more suitable for implementation in a functional language
    - Used in Haskell rather than the pure mathematics form of Mac Lane
- Beck's Theorem
  - Provides rules on which categorial transformations in the T-algebra $X \rightarrow X^T$ are valid.
    - Sometimes called PTT (Precise Tripleability Theorem)

Cartesian Monads

- If underlying categories are pullbacks
  - AND T preserves pullbacks
  - AND μ and η are Cartesian
- Then the monad is Cartesian
  - Facilitates its use in transformations where a Cartesian type is expected

Summary of Progress

- Topos has been established as data-type of choice
- Monad shows potential for processing the topos and for transforming the topos

- Areas for attention:
  - Intension/extension in topos, including pullbacks, subobject classifier and operations by the monad
  - Exploring usefulness of additional work on monads including those mentioned here: T-algebra, Kleisli, Beck and Cartesian monad