Monadic Design for Universal Systems

Nick Rossiter
Visiting Fellow
Computing Science and Digital Technologies
Northumbria University

ANPA 37 (August 2016, modified March 2017)

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	ANPA 36: 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	ANPA 36: Expressed in complex, more realistic design, satisfies 5NF
Recursive pullbacks	ANPA 35: Represent different levels of detail
AllegoriesANPA 36:	ANPA 36: Explored, not useful in natural IS but significant for interoperability

Normalization

- Match in data design between logical and physical world.
- Many stages of normal forms in set-theoretic relational model:1NF, 2NF, 3NF, BCNF, 4NF, 5NF.
- Only last need concern us here.
- 5NF is also known as Project-Join Normal Form

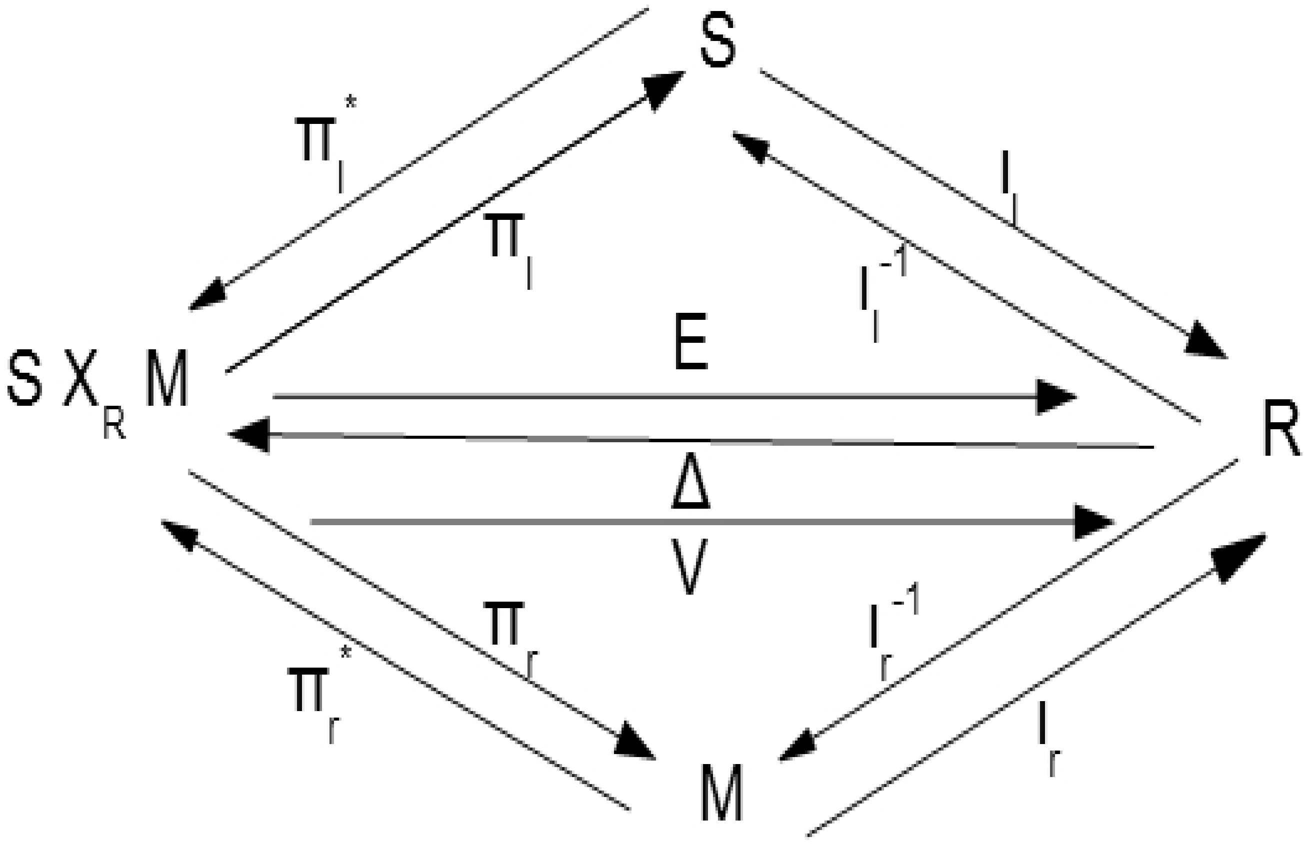
5NF (PJNF) is a Pullback Diagram

- Structure is in 5NF if its projections can be joined to return the original structure.
- Projection is decomposition of a product
- Join is Δ, diagonal from coproduct to product
- Mark Levene & Millist W Vincent evidently consider pullbacks with normalisation
- Simple pullback (two projections, one join) is trivial
- More complex (realistic) examples are handled by pasted pullbacks

Examples

- Student Marks
  - Simple PJ/NF (single pullback)
- Bank Transactions
  - Simple PJ/NF (single pullback)
  - Simple pasted PJ/NF (2 pasted squares, 3 pullbacks)
  - Complex PJ/NF (5 pasted squares, 10 pullbacks)
  - Complex structure (5 pasted squares, not valid pullback)

Pullback - Single Relationship
Student Marks



Pullback - Single Relationship Constraints

- SX_R M (Student X_{Result} Mark)
- Logic of adjointness: $E \dashv \Delta \dashv V$
  - Δ selects pairs of S and M in a relationship in context of R
  - Such that $E \dashv \Delta$ and $\Delta \dashv V$
  - Termed by Lawvere as a hyperdoctrine
- Projections π are from product, left and right (dual π^*)
- Inclusions $\imath$ are into sum S+M+R, left and right (dual $\imath^{-1}$)
- S, M, R are categories, with internal pullback structure, giving recursive pullbacks

Recursive Pullbacks

A node of a pullback may itself be a pullback

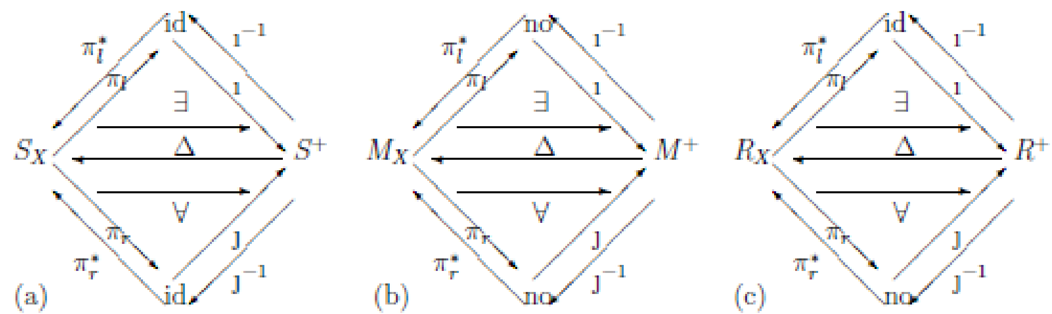
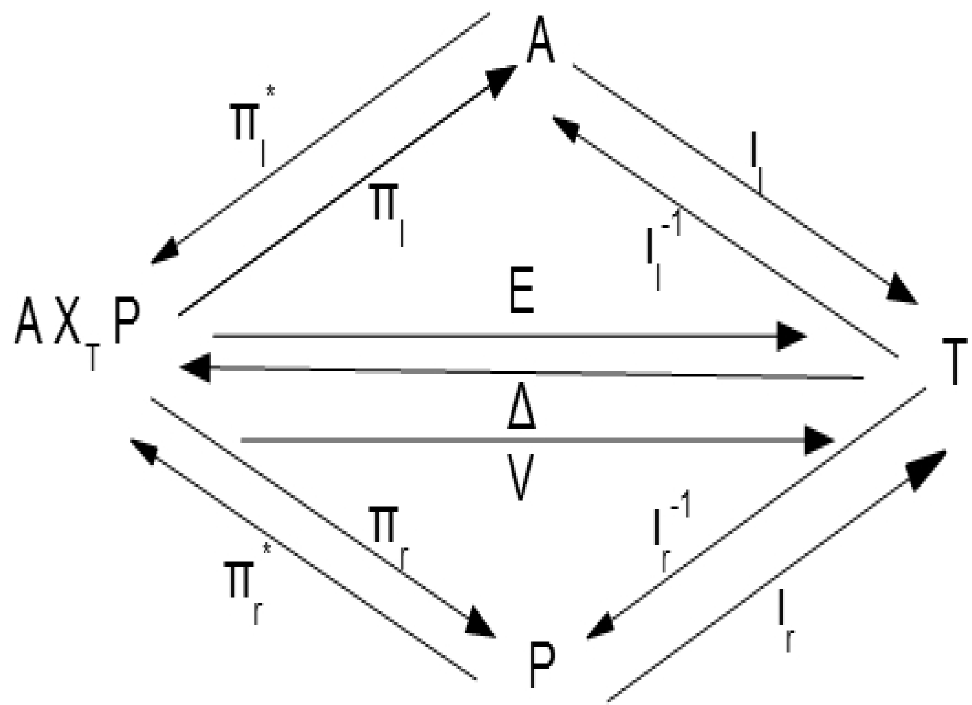


Figure 13: Internal Structure of Categories: a) The Pullback in **S**. S_X is id $\times_{S+}$ id, S^+ is name $+_{\text{id}}$ address. b) The Pullback in **M**. M_X is no $\times_{M+}$ no, M^+ is title $+_{\text{no}}$ grade, c) The Pullback in **R**. R_X is id $\times_{O+}$ no, R^+ is mark $+_{\text{id+no}}$ decision.

Each node in the pullback for Student over Marks in context of Result is itself a pullback, giving a recursive structure

Pullback - Single Relationship: Bank Transactions by Account and Process

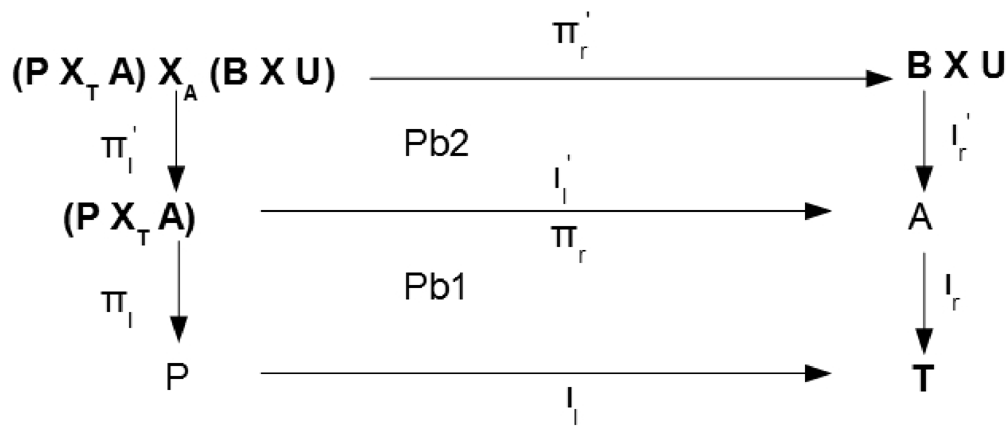


Pullback - Single Relationship Details

- $A \; X_T \; P$ (Account $X_{Transaction}$ Process)
  - Account can belong to many users
  - Process is type of transaction: e.g. standing order, direct debit, ATM cash withdrawal
  - Transaction is item for transfer of funds according to ACID requirements
- A, P, T are categories, with internal pullback structure, giving recursive pullbacks

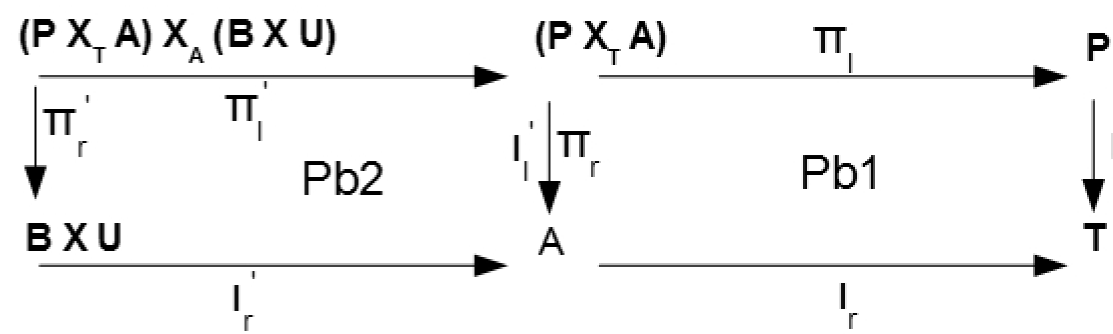
Pullback - Two Pasted Relationships: Bank Transactions by Branch/User

Three Pullbacks Pb1, Pb2, Pb2 X Pb1



B is Branch, U is user, B X U as user can have multiple accounts across the banking network

Usually written in horizontal (landscape) form. Vertical layout enables deep nested structures to be represented more readily

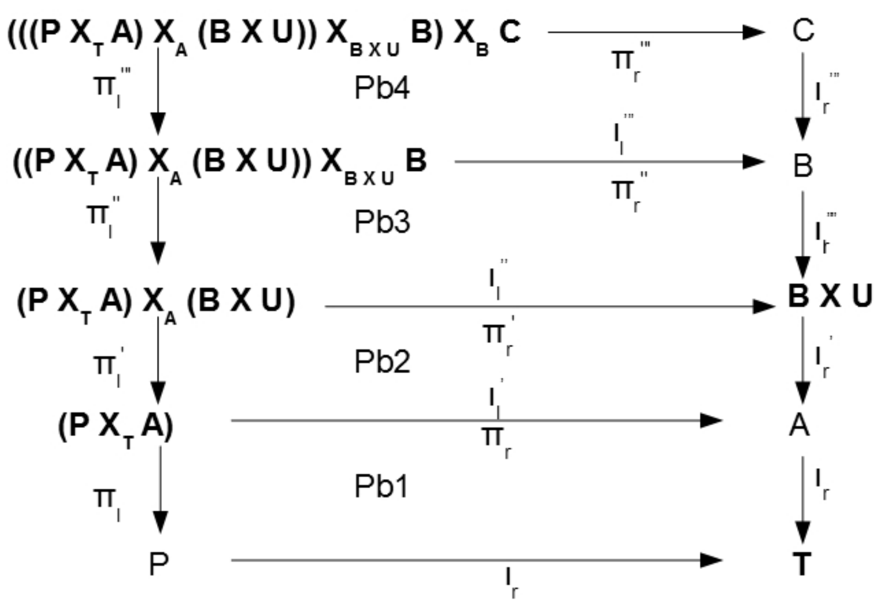


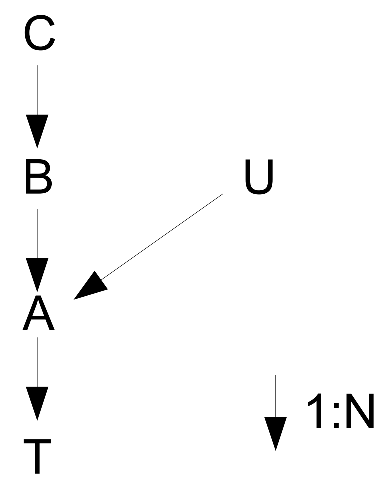
Pasting condition for Pb2 X Pb1: $\iota_l' = \pi_r$ after Freyd's Pasting Lemma

For our purposes, a pasted pullback is only a valid pullback if all inner and outer diagrams are pullbacks

Pasting is associative (order of evaluation is immaterial) but not commutative (relationship A:B 1:N is not same as A:B N:1)

Pullback – x10 Natural Bank Account Transactions





C company, B branch, U user, A account, P process, T transaction

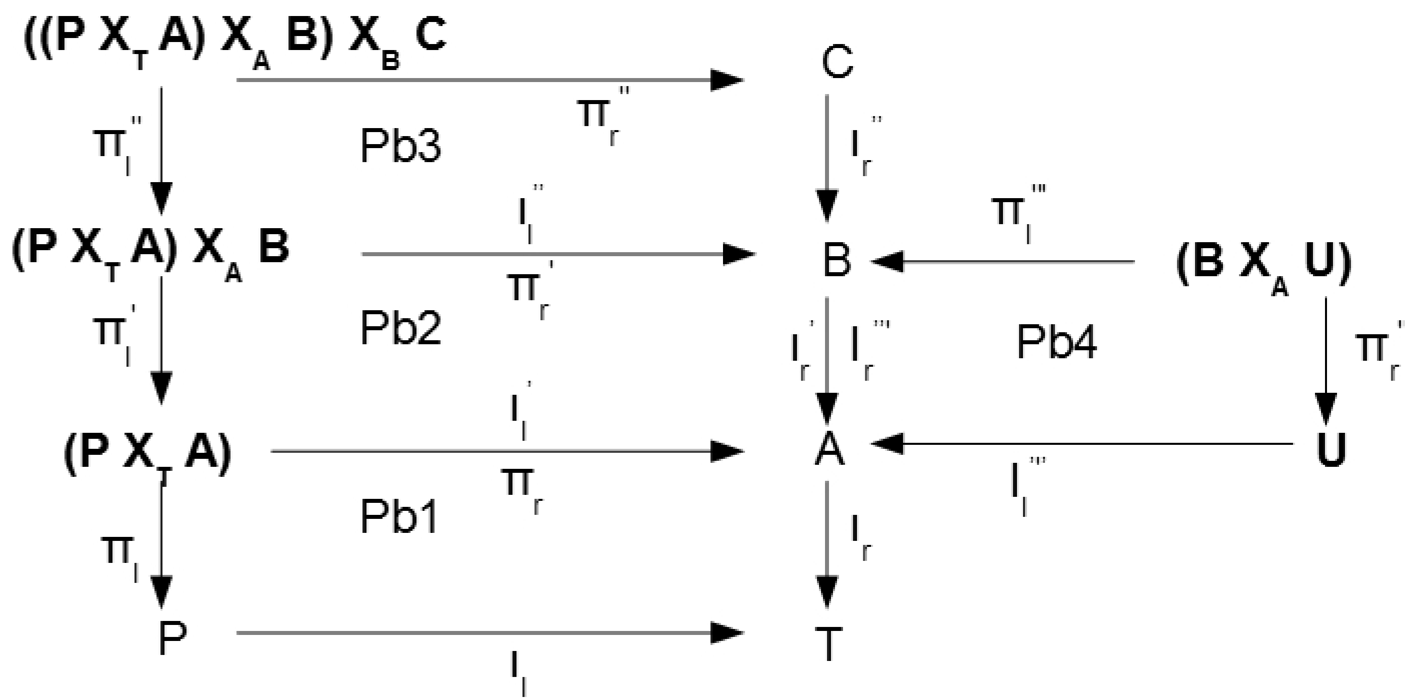
10 pullbacks: Pb1, Pb2, Pb3, Pb4
Pb2 X Pb1, Pb3 X Pb2, Pb4 X Pb2
Pb3 X Pb2 X Pb1, Pb4 X Pb3 X Pb2
Pb4 X Pb3 X Pb2 X Pb1

For our purposes, a pasted pullback is only a valid pullback if all inner and outer diagrams are pullbacks

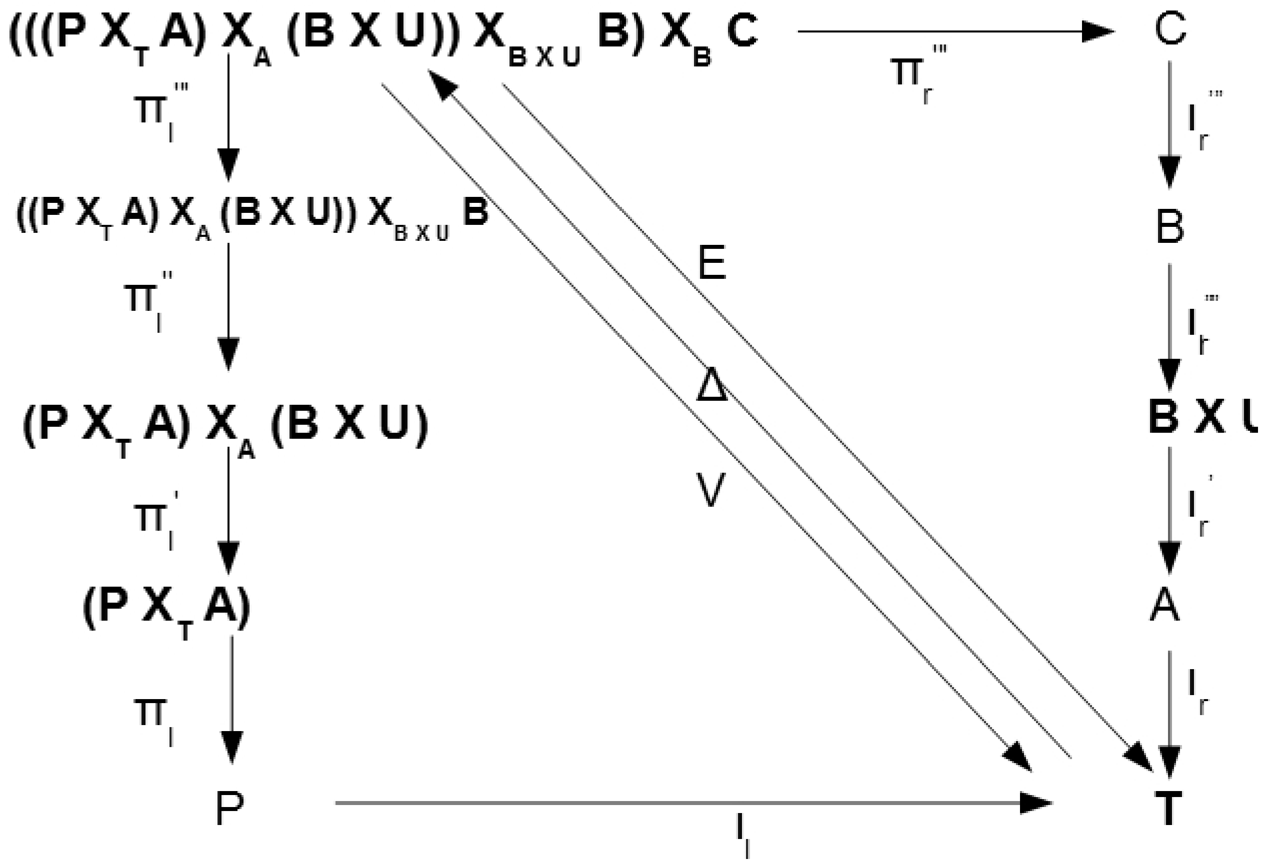
Invalid Pullback

Invalid as
not all squares
are pullbacks

For instance
Pb4 X Pb2 is not
a pullback



Adjointness Holds for all Pullbacks



E ⊣ Δ ⊣ V for this outer pullback and all other 9 pullbacks

Pasting Pullbacks – Discussion 1

- All diagrams commute
- All diagrams, inner or outer, are pullbacks
  - In pure maths, the condition is relaxed a little
    - Not appropriate for applied
- Structure is recursive
  - A pullback node may be a pullback structure in its own right
  - No limit to recursion

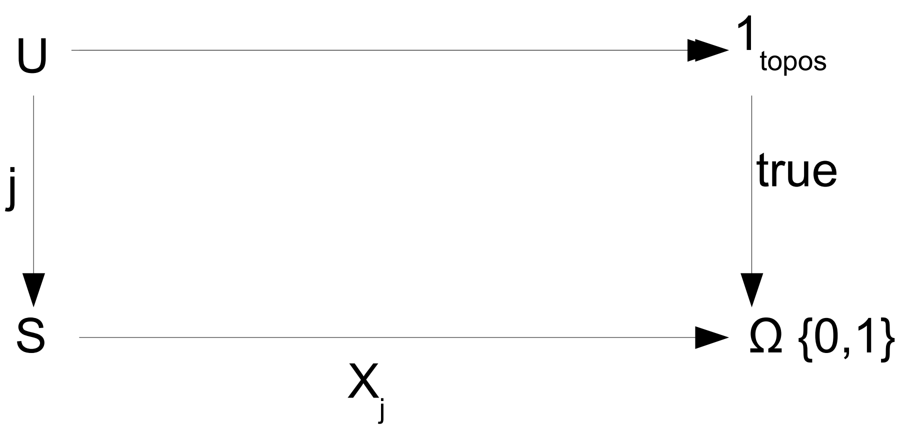
Pasting Pullbacks – Discussion 2

- Meets all Information System requirements as in 5NF (Project-Join)
  - Pullback is project-join through Π and Δ
  - Not categorification of set-theoretic approach
  - 5NF was a belated move by set-theoretic adherents to find a viable approach, which happened to follow category theory principles
- Pasting condition appears to be:
  - $\iota_l' = \pi_r$ (left-inclusion of outer square = right-projection of inner square)
  - Discussed further later

Pasting Pullbacks – Discussion 3

- Pasted structure
  - is a Cartesian Closed Category (CCC) with products, terminal object and exponentials
  - is a topos as a CCC with subobject classifier and internal Heyting Logic
- The subobject classifier provides an internal query language

Subobject Classifier – Searching within a Topos - Boolean example



Simple database query in category theory style

Ω {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

The Pasting Condition 1

- $\iota_l' = \pi_r$ (left-inclusion of outer square = right-projection of inner square
  - Looks rather set theoretic
- But any pullback can be represented as an equalizer (ncatlab)

$$A \times_T P \longrightarrow A \times P \underset{\iota_r \pi_r}{\overset{\iota_l \pi_l}{\rightrightarrows}} T$$

The Pasting Condition 2

Similarly for a pasted pullback, the equaliser is

$$(P\ X_T\ A)\ X_A\ (B\ X\ U) \longrightarrow (P\ X_T\ A)\ X\ (B\ X\ U) \overset{I_l \pi_l \pi'_l}{\underset{I_r \pi_r \pi'_r}{\rightrightarrows}} A$$

Equals in sets is undefined as context is not defined

Equaliser in categories, as a limit, is fully defined up to natural isomorphism

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

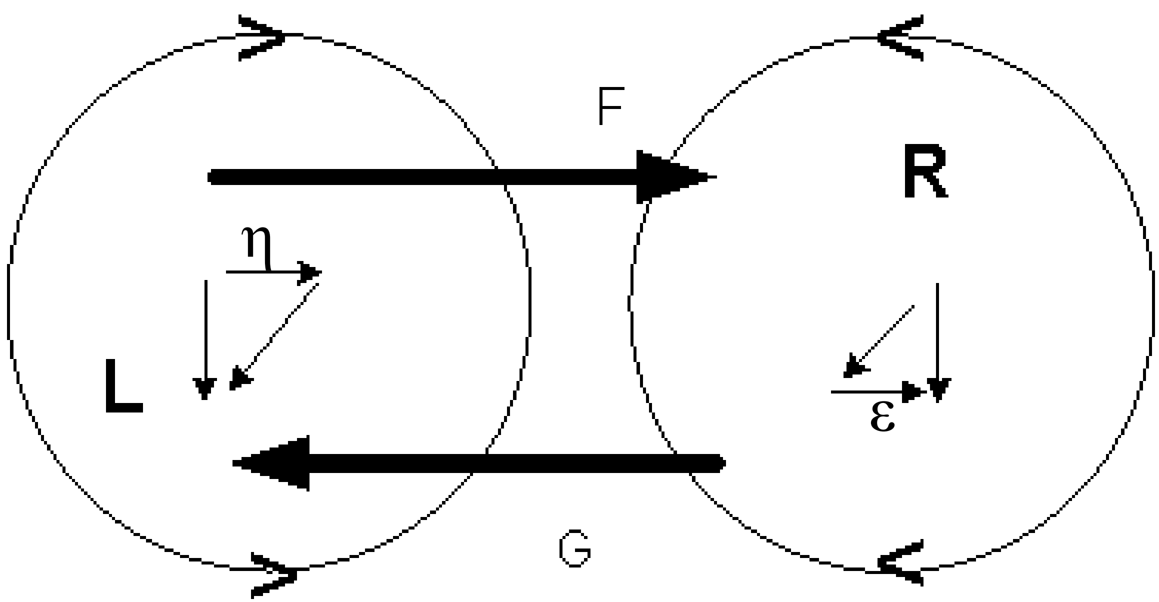
- Principles of ACID
  - Atomicity, Consistency, Isolation, Durability
- Logical technique for controlling the physical world
  - e.g. banking transaction
- Requires three cycles of adjointness between initial and target state
- First two for atomicity, consistency and isolation
  - First makes changes; 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 -| 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 → X, functor with same source and target)
  - 3 is the unit of adjunction η: 1_X → T (change, looking forward)
  - 4 is the multiplication μ: T X T → 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?
- $\langle X, T, \eta, \mu \rangle$ is reduced to
  - $\langle T, \eta, \mu \rangle$ 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

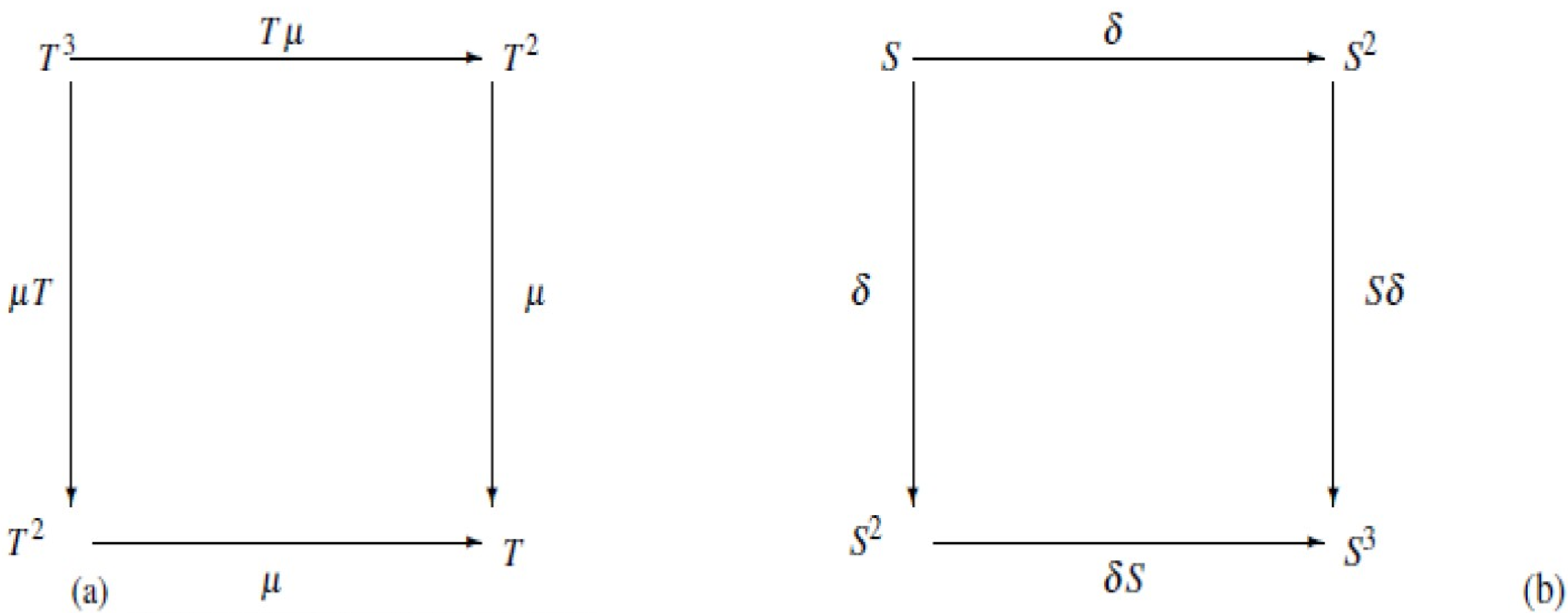
- The dual of the monad
- A comonad is a 4-cell <1,2,3,4>
  - 1 is a category X
  - 2 is an endofunctor (S: X → X, functor with same source and target, S is dual of T)
  - 3 is the counit of adjunction ε: S → 1_X (change, looking back)
  - 4 is the comultiplication δ: S → S X S (change, looking forward)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

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:



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

Unitary Law for Monads

- The diagram commutes

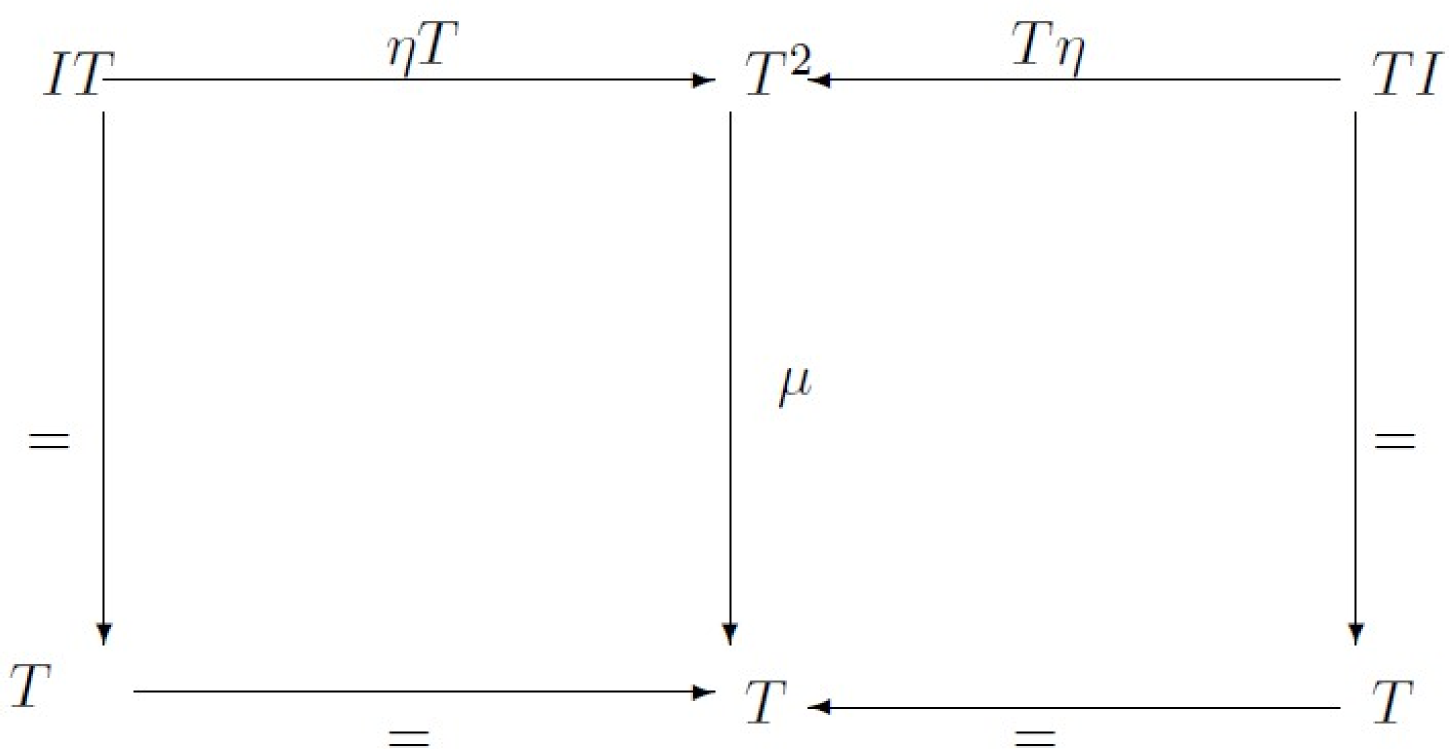


Figure 13: Left and Right Unitary Laws for Monad $T =< T, \eta, \mu >$

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 $\langle T, \eta, \mu \rangle$ is called the monad T
- Similarly for the comonad
  - $\langle S, \varepsilon, \delta \rangle$ is called the comonad S
- It's a synecdoche

Operating on a Topos

- The operation is simple:
  - T: E → E
    - where T is the monad <GF, η, GεF> 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

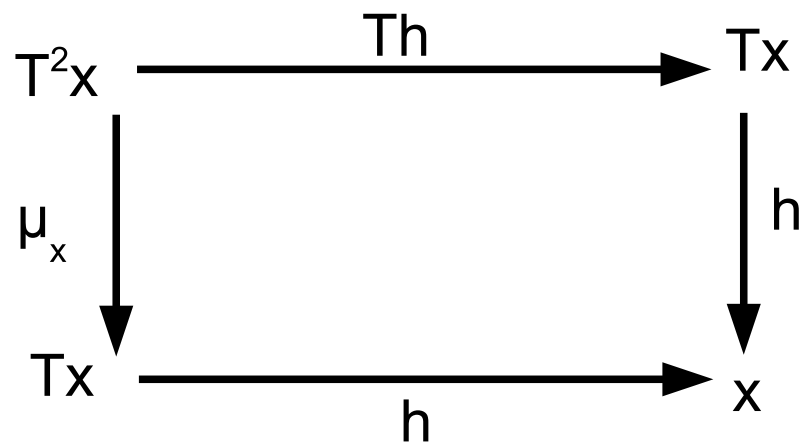
- For a category X, not necessarily a topos, in the monad <X, T, η, μ>, the effect is to obtain:
  - $<G^T F^T, \eta^T, G^T \varepsilon^T F^T>: X \rightarrow X^T$
- That is a new monad adjunction $F^T \dashv G^T$ is defined to accommodate the changed category X^T
- For a topos E, this is equivalent to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

The T-algebra

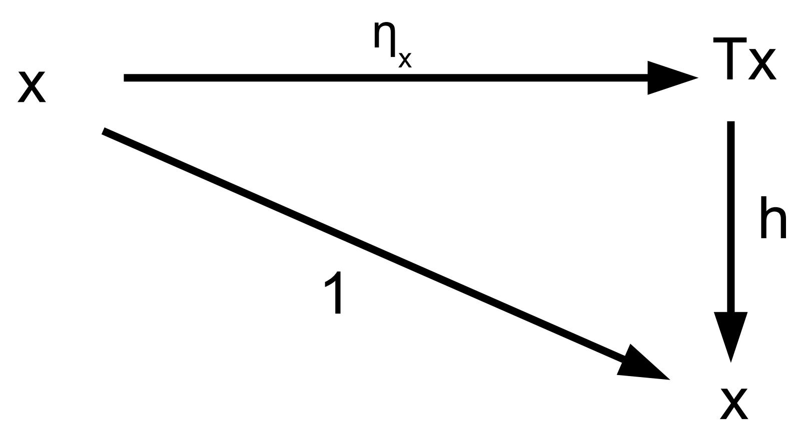
- For a monad <T, η, μ> in X
- A T-algebra is:
  - <x, h>
- Where x is an an object in X
- And h: Tx → x is the structure map of the algebra
- Such that the following diagrams commute.

T-algebra: Associative/Unitary Laws

Associative



Unitary



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