The Monad in Process-Relational Systems

Nick Rossiter & Michael Heather
Department of Computer Science and Digital Technologies
Northumbria University, NE1 8ST, UK
nick.rossiter1@btinternet.com; michael.heather@trinity.cantab.net
http://www.nickrossiter.org/process/

Abstract

The work described here builds on recent work presented on structure and process in the universe, inspired by the work of Whitehead in *Process & Reality*. We develop a formal attempt, using the topos and the monad, to replace the informal categories of Whitehead. The internal structure of the topos is explored with particular emphasis on the nature of the pasted pullback, including the conditions for a pasting to be valid and the inherent recursive nature of pullback structures. A banking example is explored, leading to the nature of the external processes acting upon the topos such as transactions. These processes are represented by monads, giving a three-level closure on the activity. The nature of monads is explored. The T-algebra enables changes to be made in the monad structure, giving the potential for adaptability. Monads, that have been strengthened by the Kleisli lift to the Cartesian form, can be composed naturally, facilitating the construction of large-scale information systems with reliability, as required for transactions in the banking world. The operation of the monad on the topos is consistent with the process-relational philosophy.

1 Introduction

We take a metaphysical approach to information systems based on the process-relational philosophy, inspired by the work of Whitehead in *Process & Reality* (Whitehead, 1929). The process-relational philosophy considers that the

world can be thought of a collection of interrelated processes, rejecting the Cartesian dualism of Descartes, and favouring the dynamic process (flux) of Heraclitus. The philosophy has been established by a number of workers, particularly in the area of social interaction. A contemporary of Whitehead, Mary Follett, was working on the idea of a social community being viewed as a process (Follett, 1919). Later workers such as Margaret Stout developed an ontology, based on Follett's administrative theory, described as a process philosophy (Stout & Staton, 2011). An associated development was process theology, as espoused by Robert Mesle (Mesle, 1993). Margaret Stout developed a philosophy, applicable in a social context, to handle creativity, Becoming, imagination and experience, extendable in a language context to ontology or Being (Stout & Love, 2015).

Much of Follett's philosophy is relevant to current requirements in computer science and information systems. There are though two problems in realising its potential. Firstly, Follett concentrated on the social side while her contemporary Whitehead was working on the physical side, in particular in metaphysics. Secondly the emphasis in information systems has been on set theory, which provides adequately the static (Being) but is restricted to process as function. Functions provide simple mappings but lack the ability to convert the logical types across the static and process components in an integrated manner. This type of problem was a major difficulty for Russell and Whitehead in their series on set theory *Principia Mathematica* (Whitehead & Russell, 1910). A single-level approach is inadequate for the complexities of information systems.

There is though a clear way forward. Much of Whitehead's *Process & Reality* can be considered as informal category theory preceding the later developments in pure mathematics, starting in the 1940s by such workers as Eilenberg and Mac Lane (Eilenberg & Moore, 1965) and resulting in what we term EML category theory in the 1990s (Mac Lane, 1998). For instance Whitehead's category of prehension, or grasping, corresponds to the categorial adjunction. Other examples are that Whitehead's category of the ultimate corresponds to the topos and his category of existence to the Cartesian Closed Category. So by combining Whitehead's metaphysics and Mac Lane's category theory, we can realise the process-relational philosophy in theory, and in applications through implementations of category theory in functional languages. In more detail we consider how the process-relational philosophy, naturally arising from *Process & Reality*, can be considered formally in category theory by the monad, which through process relates inputs

and outputs as an adjunction. The monad operates on a category, such as a topos, over three-levels, providing the necessary closure of being defined as unique up to natural isomorphism. The term monad is very 'old' but was made better known by Leibniz. We have made a comparison of the various usages of the term, including its use today in mathematics and computer science.

The fundamental categorical facilities identified for a Universe, whether from any Universe of Discourse up to the Universe, are the Topos as a structural data-type and the monad as a process. The application of the monad to a topos gives the operation of a process on data at the highest level, defined as a unique solution up to natural isomorphism. We will demonstrate such an application and explore its potential. The topos is a fundamental Cartesian Closed Category (CCC), a category with limits and exponentials subobjects, closed at the top with the terminal object. A CCC has an internal logic of the typed λ-calculus, an identity functor and the interchangeability of levels, with nodes being either objects or categories. A topos has additional properties beyond a CCC ((Mac Lane, 1998), at p.106) including a subobject classifier, the internal logic of Heyting, that is intuitionistic logic, and a reflective subtopos category for recursive query closure. The Heyting lattice is modelled in set theory by the typed $\lambda-$calculus.

The application of the topos to data was established in our earlier work introducing the topos/monad approach (Rossiter, & Heather, 2015) and bringing out the interoperable use of allegories for legacy relational systems (Rossiter, & Heather, 2016). Structures developed as a topos include pasted pullbacks, to represent relatedness, and recursion in which any juncture in the structure is a pullback in its own right. The exact nature of the match, in the pasting operation, is discussed later. Data normalisation is the standard technique for evaluating a data design, in particular to determine how closely the logical design matches the physical world. A number of stages have been developed for the set-theoretic relational model: 1NF (First Normal Form), 2NF, 3NF, BCNF, 4NF, 5NF. The last and most demanding stage 5NF concerns us here, not just for its rigour but for its definition in category theory terms, indicated by its alternative name of Project-Join Normal Form (PJNF). In set theoretic terms, the definition of 5NF is that the structures resulting from the projections can be joined together to return the original structure without loss or gain of information (Kent, 1983). Other less powerful normalisation techniques are considered to be so set-based that any categorial approach would be categorification. The Cocartesian dual to the

topos may offer further insights into the data structuring process; a structure that is closed at both the top and the bottom suffers from the closed world assumption (CWA).

The use of the allegories of Freyd (Freyd & Scedrov, 1990) as a basis for data structures was attempted (Rossiter, & Heather, 2016) but rejected because of their lack of naturality as set-based relations; the allegories will have use though in interoperability as a wrapper for legacy relational databases. Internal queries on a topos are handled by the subobject classifier, which may be Boolean (0 or 1) or the more general double powerobject. Both forms were illustrated in an earlier paper (Rossiter, & Heather, 2016). The provision of examples of Heyting intuitionistic logic for an application remains an objective. Internal queries are more akin to data searches, such as through Google, but do not provide a well-defined process capability.

The first application used was of student marks in a university context, which was adequate from the data structure viewpoint but limited from a data process angle. A more interesting application from the process perspective is banking, including the handling of transactions. This was first studied by us in ANPA 27 (Rossiter, Heather, & Sisiaridis, 2006).

Monadic design is a novel technique for handling the dynamic aspects of an application. Aspects to be investigated are the adjointness, inherent in the approach, the flavours of monad which are most suited to process applications and the T-algebra for modifying the adjunction. The intention in this paper is therefore to introduce a further application, banking, which provides a more suitable test for an external process of a monad on a topos data structure. The mechanism of pasting is to be investigated in detail and the relationship of the topos to database normalisation is to be clarified. Monadic design will be developed for the topos.

2 Pullback: Single Relationship

The topos has limits and the pullback is a limit. Figure 1 shows for the student application, introduced in (Rossiter, & Heather, 2015), a simple pullback of assessment : $\mathbf{S} \times_{\mathbf{R}} \mathbf{M}$, the product of Student and Mark in the context of Result. The relationship between the product $\mathbf{S} \times_{\mathbf{R}} \mathbf{M}$ and $\mathbf{R}$ is adjoint, with the following logic condition holding: $\exists \dashv \Delta \dashv \forall$. The functor Δ selects pairs of $\mathbf{S}$ and $\mathbf{M}$ in a relationship in the context of $\mathbf{R}$, such that $\exists$ is left adjoint to Δ and $\forall$ is right adjoint to Δ, with a consequential facility for

consistent logical operation. A diagram with such adjointness was termed by Lawvere as a hyperdoctrine in the early days of EML category theory (Lawvere, 1969). The existential functor $\exists$ records the decision for each student (under the free functor F) of a specific mark. The diagonal Δ sorts student and marks as a component of the underlying functor G. The universal functor $\forall$ produces the final mark list generated by F. This shows the fine structure of the adjointness $F \dashv G$. Other arrows are interpreted as follows. Projections π are from the product onto its constituents, left π_l and right π_r, with dual arrows left π_l^* for student capability and achievement and right π_r^* for quality of work respectively. Inclusions ι are into the sum $\mathbf{S} + \mathbf{M} + \mathbf{R}$ from its constituents, left ι_l for candidature and right ι_r for marking, with dual arrows ι_l^{-1} and ι_r^{-1} respectively.

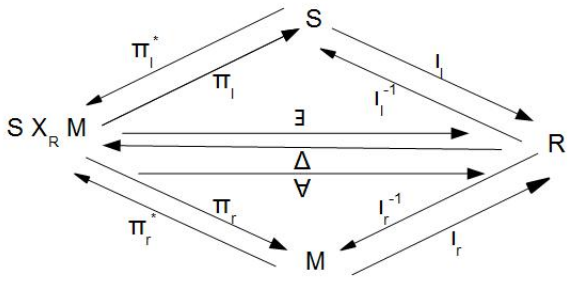

Figure 1: Pullback for a Single Relationship $\mathbf{S} \times_{\mathbf{R}} \mathbf{M}$; $\mathbf{S}$ Student, $\mathbf{M}$ marks, $\mathbf{R}$ Result

$\mathbf{S}, \mathbf{M}, \mathbf{R}$ are each categories, with an optional internal pullback structure, giving a recursive pullback structure with potential unlimited depth, as shown in Figure 2. These diagrams show the objects present in each of the categories with the potential for each of the objects to be itself a category, as in a recursive structure. The categories shown at this level, the bottom of the data structure, are of the Dolittle type with a mapping from the data in the left-hand pullback object, a product, to equivalent data in the right-hand pushout object, a coproduct (Heather, 2005). Such diagrams are also called Bicartesian squares (Banach, 1994) or pulation squares ((Adámek, 2005) pp.205-206). The top and bottom objects are apparently the same, being the identifier for the data structure. However, we agree with Lambek & Scott ((Lambek & Scott, 1986) pp.65-68) that they are different in purpose,

with the top object being the intension (definition) and the bottom object the extension (values). Everything is related implicitly in a + context; the relationship in a × context is stronger with explicit connections. Relatedness in Heyting logic is expressed by the condition: $C \times A \leq B$ is isomorphic with $C \leq A \Rightarrow B$. Such a view of relatedness is not available in set theory.

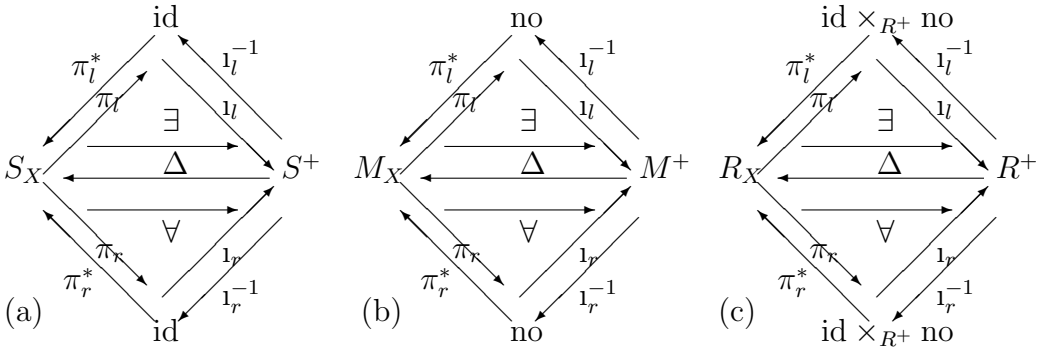

Figure 2: Internal Structure of Categories: a) The Pullback in **S**. S_X is $\mathrm{id} \times_{S^+} \mathrm{id}$, S^+ is $\mathrm{name} +_{S_X} \mathrm{address}$. b) The Pullback in **M**. M_X is $\mathrm{no} \times_{M^+} \mathrm{no}$, M^+ is $\mathrm{title} +_{M_X} \mathrm{date}$, c) The Pullback in **R**. R_X is $(\mathrm{id} \times_{R^+} \mathrm{no}) \times_{R^+} (\mathrm{id} \times_{R^+} \mathrm{no})$, R^+ is $\mathrm{decision} +_{R_X} \mathrm{board}$.

Figure 2(a) gives the full candidate record with, for each student, *id* being their identifier or key and *name* and *address* being other details held. S_X, an explicit relationship, is the product of the intensional *id* with the extensional *id* in the context of S^+. The *id* may be typed by name or as a surrogate number; a number is more appropriate for anonymous marking. The coproduct S^+, an implicit relationship, is the sum of the intensional key, the extensional key and the remaining attributes with their intensional/extensional values, all in the context of S_X. The arrow $\exists$ picks out a student's details from their id; the arrow $\forall$ picks out details for all students and Δ establishes the identity for a student from their records. The arrows π_l and π_r are projection arrows from the product S_X to the intensional and extensional identifiers respectively. The arrows ι_l and ι_r are inclusion arrows from the intensional and extensional identifiers respectively to S^+. All the family of π and ι arrows are deterministic (1:1). In general pullback diagrams, the π and ι arrows express properties of the relationship. Here, with diagrams of the Dolittle type, such arrows are always 1:1 – an entry in the product and coproduct is always associated with one entry in the intension and extension, and *vice*

versa.

Figure 2(b) gives the full marks record. For each mark, *no* is the mark as an integer, *title* is the module title, *date* is the date of the assessment. M_X, an explicit relationship, is the product of the intensional *no* with the extensional *no* in the context of M^+. M^+ is the sum of the intensional key, the extensional key and the remaining attributes with their intensional/extensional values, all in the context of M_X. The product M_X is the classification system with π_r giving a list of classifiers in the extension. The arrow $\exists$ picks out the use of a mark from a particular *no*, the arrow $\forall$ picks out details for all *no* used and Δ establishes a mark awarded from the marking records.

Figure 2(c) gives the full results record. For each result, *id* and *no* have already been defined, *decision* is the final grade as awarded by the *board*. The combination of *id* and *no* forms the key R_X as $(\text{id} \times_{R+} \text{no}) \times_{R+} (\text{id} \times_{R+} \text{no})$, representing the product of intensional $\text{id} \times_{R+} \text{no}$ and extensional $\text{id} \times_{R+} \text{no}$ in the context of R^+. The coproduct R^+ is $((\text{mark} +_{R_X} \text{no}) +_{R_X} \text{decision})$, the sum of the intensional key, the extensional key and the remaining attributes with their intensional/extensional values, in the context of R_X. With two attributes as the key, we have maintained the same Dolittle structure with the top and bottom attributes being the intension and extension respectively and the family of π and ι arrows all being deterministic. The arrow $\exists$ picks out the details of a result for a particular student and mark; the arrow $\forall$ picks out details for all student-mark combination and Δ establishes a valid student-mark pair from the records of results.

3 Banking Examples

3.1 Pullback: Single Relationship

We now introduce the Banking example, which is a more suitable subject for illustrating the action of process on a topos. The simple pullback is shown in Figure 3, defined as $\mathbf{P} \times_{\mathbf{T}} \mathbf{A}$, that is the product of Procedure and Account in the context of Transaction, with $\mathbf{P}$ the category Procedure, $\mathbf{A}$ the category Account, and $\mathbf{T}$ the category Transactions. An Account can belong to many users; the Procedure is the type of the transaction, for example: standing order, direct debit, ATM cash withdrawal; the transaction is a transfer of funds according to data processing requirements. $\mathbf{P}, \mathbf{A}, \mathbf{T}$ are categories, with internal pullback structure, giving recursive pullbacks as required, as in

Figure 2 for the student example.

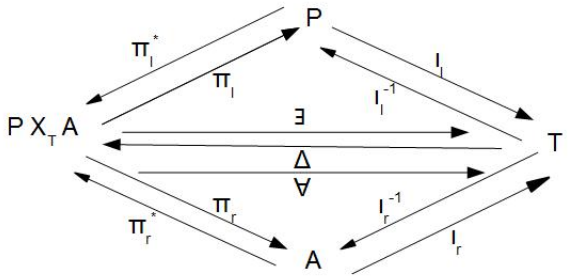

Figure 3: Pullback - Single Relationship: Bank Transactions **T** by Procedure **P** and Account **A**

3.2 Pullback: Two Pasted Relationships

In pasted pullbacks two relations are joined together to form a square. An additional category is introduced for User (customer) of **U**. Each user may have multiple accounts across the banking network: there is a many-to-many (N:M) relationship between **U** and **A**. The second pullback is the product of the subproduct of the first pullback $\mathbf{P} \times_{\mathbf{T}} \mathbf{A}$ with **U** in the context of **A**, as shown in Figure 4. The resulting relationship is of account transactions by user. For the purpose of discussion, the pullbacks can be labelled $Pb1$ for the first square $\mathbf{P} \times_{\mathbf{T}} \mathbf{A}$ and $Pb2$ for the second square $(\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times_{\mathbf{A}} \mathbf{U})$. By standard EML category theory ((Mac Lane, 1998) pp.71-72) if the squares $Pb1$ and $Pb2$ are valid pullbacks, then the whole outer square is also a pullback $Pb2 \times Pb1$. We therefore have three pullback diagrams in a valid pasted relationship.

The vertical stacking of the pasted pullbacks, one above the other, in portrait form is suited to practical applications which could involve 5-10 relationships in a deep nested structure. In category theory text books, pasted structures are usually written in horizontal (landscape) form as in Figure 5, which is logically identical to that in Figure 4.

The aim of pasting in topology is to 'glue together' two continuous functions to create another continuous function. The specific pasting condition

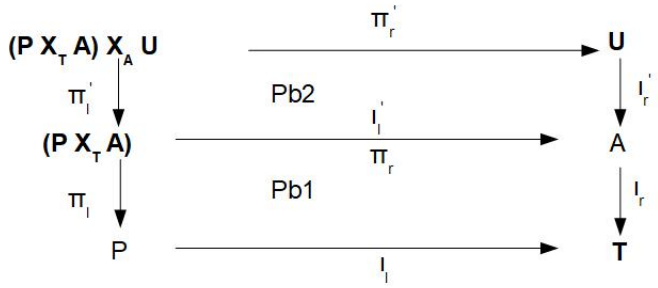

Figure 4: Pullback: Two Pasted Relationships: Bank Transactions by User, in Portrait Layout

for the pullback $Pb2 \times Pb1$ is that $\iota_l' = \pi_r$ after Freyd's Pasting Lemma (Freyd & Scedrov, 1990).

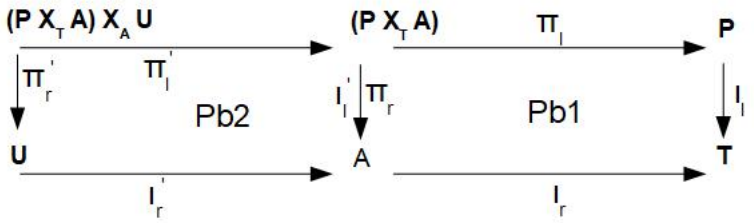

Figure 5: Pullback: Two Pasted Relationships: Bank Transactions by User, in Conventional Landscape Layout

To make the application more realistic we add two further categories, those of **B** for Branch and **C** for (banking) Company. Branch:User is also a N:M relationship as each Branch has many Users and each User has many Branches but Company:Branch is a 1:N relationship: each Company has many Branches, each Branch is within one Company. The overall relationship is $(((\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times_{\mathbf{A}} \mathbf{U}) \times_{\mathbf{U}} \mathbf{B})$ with **C** in the context of **B** giving the pullback diagram shown in Figure 6. The representation of N:M and 1:N relationships is the same in terms of pullback structures, giving a useful symmetry in data design.

Figure 6 involves six categories: **C** company, **B** branch, **U** user, **A** ac-

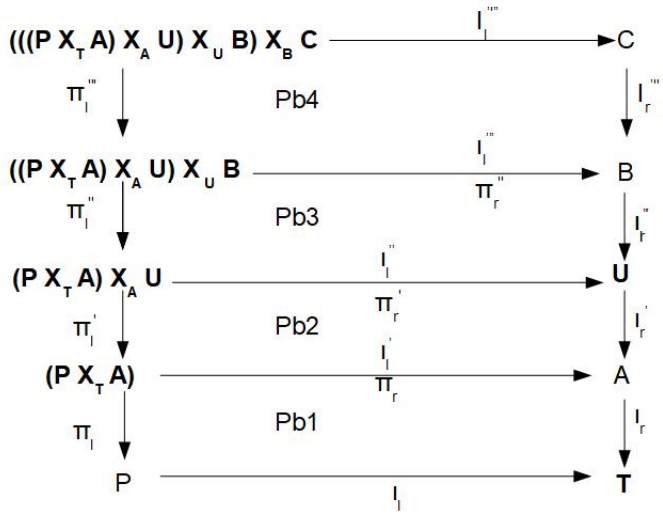

Figure 6: Pullback: Three Pasted Relationships: Bank Transactions by User by Company Branch

count, **P** procedure, **T** transaction, and ten pullbacks: $Pb4, Pb3, Pb2, Pb1$; $Pb4 \times Pb3, Pb3 \times Pb2, Pb2 \times Pb1$; $Pb4 \times Pb3 \times Pb2, Pb3 \times Pb2 \times Pb1$, $Pb4 \times Pb3 \times Pb2 \times Pb1$. The relations within a banking system are shown in more conventional form in Figure 7(a) where each single-headed arrow represents a 1:N (one-to-many) relationship and each double-headed arrow represents a N:M (many-to-many) relationship.

For our purposes, a pasted pullback is only a valid pullback if all inner and outer diagrams are pullbacks. There are some theorems in EML category theory ((Mac Lane, 1998) pp.71-72) which enable some deductions to be made based on partial knowledge: for example, with the diagram in Figure 5, if the inner diagrams are pullbacks then the outer diagram is a pullback, as stated earlier, and if the outer diagram and the right-hand diagram are pullbacks then the left-hand diagram is a pullback. Such deductions could be facilitated in any practical system but are a distraction from developing a simple robust solution.

As an example of an invalid pullback, consider the diagram in Figure 8 where the relationship diagram has been modified to that in Figure 7(b). There are seven valid pullbacks in the diagram: $Pb4, Pb3, Pb2, Pb1$; $Pb3 \times Pb2, Pb2 \times Pb1$; $Pb3 \times Pb2 \times Pb1$, but not all squares are pullbacks, for

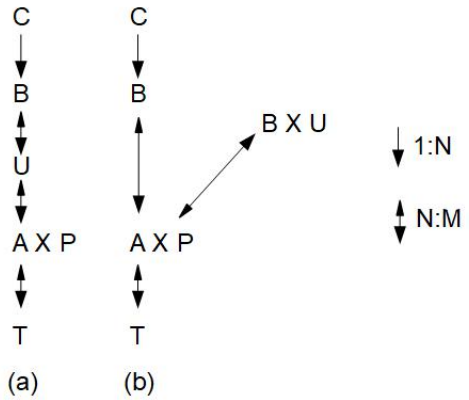

Figure 7: Relations within a Banking System corresponding to (a) Figure 6 and (b) to Figure 8. C is company, B branch, U user, A account, P procedure, T transaction.

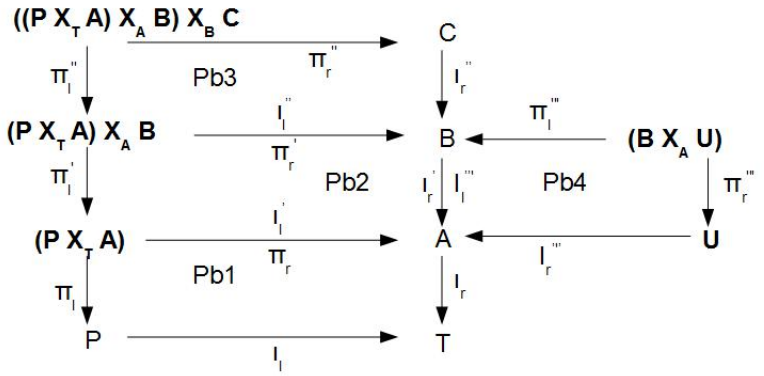

Figure 8: Invalid Pullback Diagram, corresponding to Relations in Figure 7(b)

example $Pb4 \times Pb2$. Therefore the whole diagram is not a valid pullback. For any valid pullback, the logic of adjointness holds for the outer square and all inner squares. Therefore for Figure 6 with its ten valid pullback diagrams, the logic $\exists \dashv \Delta \dashv \forall$ holds across every diagram. An example of this logic is shown in Figure 9 for the outer square.

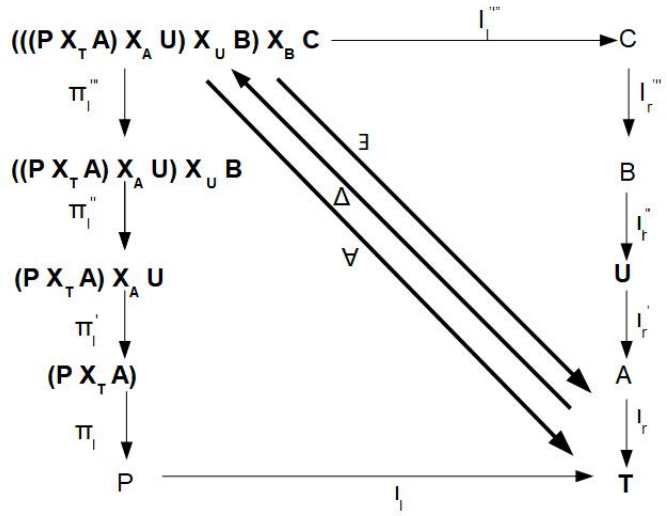

Figure 9: Adjointness Holds for all Pullbacks: $\exists \dashv \Delta \dashv \forall$

4 Pasting Pullbacks: Discussion

To summarise, in a pasted diagram, all pullbacks as inner or outer squares must commute for the diagram to be a valid pullback as a whole. The structure is recursive in that a pullback node may itself be a pullback diagram. Two aspects are worthy of further discussion: how does the pullback diagram relate to data normalisation in conventional data structuring and can the pasting condition be expressed in other forms, drawing out the nature of the '=' condition?

4.1 Normalisation

In set theoretic terms, the definition of 5NF is that the structures resulting from the projections can be joined together to return the original structure

without loss or gain of information (Kent, 1983). Looking at the simple pullback diagram, as in Figure 3, the projections are the π arrows, π_l and π_r, and the join arrow is the diagonal Δ. PJNF holds through the adjointness in every pullback: $\exists \dashv \Delta \dashv \forall$. The arrows $\exists$ and $\forall$ involve the projections through the compositions: $\exists = \iota_l \circ \pi_l = \iota_r \circ \pi_r$ and $\forall = \iota_l \circ \pi_l = \iota_r \circ \pi_r$. In more complex data structures, the same logic applies. For instance in Figure 9 with six pullback squares (including undrawn inner ones), PJNF will hold if the whole structure and all inner squares are pullbacks with the logic: $\exists \dashv \Delta \dashv \forall$. Surprisingly pullbacks have rarely been used in normalisation studies, an exception being the work of Levene & Vincent (Levene & Vincent, 2000) who briefly mention the pullback inference rule, following from the interaction between functional dependencies $\exists$ and inclusion dependencies ι.

It should be emphasised that the pullback is not categorification of the set-theoretic approach to normalisation of 5NF, as in earlier work with category theory and databases (Johnson & Rosebrugh, 2002). The form 5NF was a belated move by set-theoretic adherents to find a viable approach to normalisation after many earlier attempts had been only partially successful. The pullback follows basic category theory principles and is a *natural* choice for an effective data structure.

4.2 The Pasting Condition

The Pasting Condition is $\iota'_l = \pi_r$, that is the left-inclusion of the outer square equals the right-projection of the inner square. On the surface this looks rather set theoretic, where the '=' would be without context, but in EML category theory the '=' is defined naturally as unique up to natural isomorphism, through the adjointness inherent in the pullback category.

Moreover any pullback can be represented as an equalizer (ncatlab, 2018), as in Figure 10, which is equivalent to Figure 3. In the equalizer diagram the product of $\mathbf{P}$ and $\mathbf{A}$ in the context of $\mathbf{T}$, $\mathbf{P} \times_{\mathbf{T}} \mathbf{A}$, maps onto the product $\mathbf{P} \times \mathbf{A}$ which in turn maps onto $\mathbf{T}$ where the two paths, $\iota_l \circ \pi_l$ and $\iota_r \circ \pi_r$, converge.

Equalizer diagrams can also be constructed for pasted pullbacks, as in Figure 11, which is equivalent to Figure 4. In the equalizer diagram the product of $\mathbf{P} \times_{\mathbf{T}} \mathbf{A}$ and $\mathbf{U}$ in the context of $\mathbf{A}$, $(\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times_{\mathbf{A}} \mathbf{U}$, maps onto the product $(\mathbf{P} \times_{\mathbf{T}} \mathbf{A}) \times \mathbf{U}$ which in turn maps onto $\mathbf{T}$ where the two paths, $\iota_l \circ \pi_l \circ \pi'_l$ and $\iota_r \circ \iota'_r \circ \pi'_r$, converge.

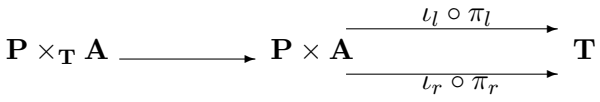

Figure 10: Pullback in Figure 3 Represented as an Equalizer

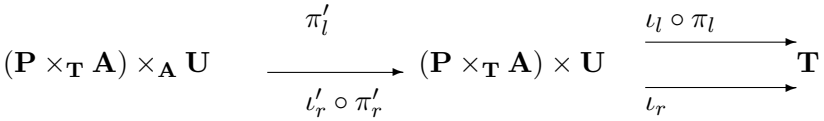

Figure 11: Pasted Pullback in Figure 4 Represented as an Equalizer

5 External Process

The concept of process is underpinned by metaphysics, as defined in the writing of authors such as Alfred North Whitehead, in his book *Process & Reality* (Whitehead, 1929). For any entity in the universe, the actions possible upon it and the rules for such actions are a critical part of the whole system. First we look at the technical features within category theory for representing process. We next look at the requirements for the real world and review the facilities of the theory that appear to be most relevant.

5.1 Process in Category Theory

An internal process is a morphism (arrow) within a topos, such as $p : A \longrightarrow B$, where the process p takes object A to object B in the same topos. Such arrows play a natural role in the category construction. An external process is activity on a topos $\mathbf{E}$, taking it to another topos $\mathbf{E}'$, such as provided by a functor F with $F : \mathbf{E} \longrightarrow \mathbf{E}'$. Both $\mathbf{E}, \mathbf{E}'$ must conform to the natural rules for topos construction. Constraints on the transition between $\mathbf{E}$ and $\mathbf{E}'$ are enforced through adjointness between F ($\mathbf{E} \longrightarrow \mathbf{E}'$) and its dual G ($\mathbf{E}' \longrightarrow \mathbf{E}$), such that $F \dashv G$ and the 4-tuple $< F, G, \eta, \epsilon >$ exists where η is the unit of adjunction $\eta : 1_E \longrightarrow GFE$, ϵ is the counit of adjunction $\epsilon : FGE' \longrightarrow 1_{E'}$ and E, E' are objects in categories $\mathbf{E}$ and $\mathbf{E}'$ respectively. The pair of adjoint functors $F \dashv G$ may be written as T and the dual $G \dashv F$ as S.

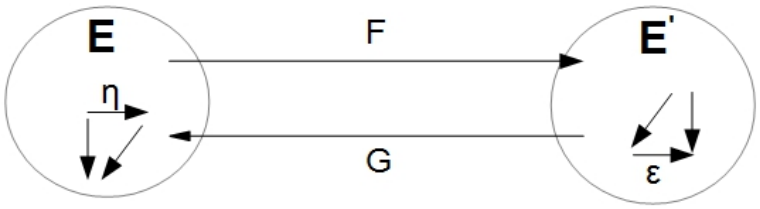

Figure 12: Multiple 'Cycles' $GFGFGF(T^3)$ for adjointness $< F, G, \eta, \epsilon >$

The cycle T can be enhanced by performing it three times, T^3, to achieve closure. Such a construction is termed a monad, with its dual S^3 a comonad. The functors and their constraints are illustrated in Figure 12. The monad is a generalisation of the single-level monoid, which has a single operation, binary multiplication $M \times M \longrightarrow M$, and the identity $1 \longrightarrow M$, for an object M.

5.2 Real-world Requirements

The process is represented in information systems by the transaction, which has been the subject of intense study because of its criticality to applications such as banking and internet-based commerce. However, the concept is a very general one, applying for instance to drafting where a transaction may last several days as a technical drawing is modified from one consistent state to another, or maybe months, as a legal document is modified similarly. The notion of transaction in a categorial context was developed in earlier ANPA papers, more generally at ANPA 31 (Heather & Rossiter, 2011), and in considerable detail at ANPA 27 (Rossiter, Heather, & Sisiaridis, 2006). The principles of the transaction are summarised as ACID: Atomicity, Consistency, Isolation, Durability. Atomicity ensures that the process, however complicated, is viewed as a single arrow. Consistency ensures that all rules have to be satisfied before the transition is made. Isolation ensures that any intermediate results in the process are not revealed. Durability ensures that

once a transaction is performed, the results persist until changed by another transaction. The transaction is a logical technique for controlling the real world.

5.3 Applicability of the Three Cycles

A transaction is viewed naturally as three 'cycles' of adjointness (Rossiter, Heather, & Sisiaridis, 2006). The first cycle performs the actual work required; the second checks for any errors or inconsistencies resulting from the first cycle; the third cycle consolidates the changes made provisionally in the first cycle and checked in the second cycle. The 'cycles' are not separate stages; all three cycles are performed as a single snap: the prehension, or grasping, of Whitehead (Whitehead, 1929). This single snap satisfies the atomicity and isolation requirements. The second cycle satisfies the consistency requirement, through review against the rules. The third cycle satisfies the durability requirement, through consolidating the results. If adjointness does not hold in any cycle, the transaction is abandoned. We now look at the application of the monad in more detail.

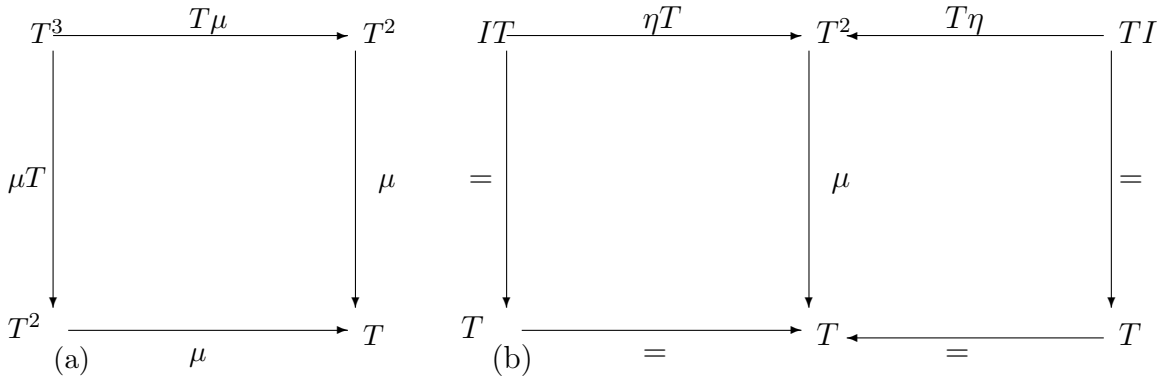

Figure 13: (a) Associative Law for Monad $< T, \eta, \mu >$; (b) Left and Right Unitary Laws for Monad $< T, \eta, \mu >$

5.4 Technical Details of the Monad Approach

For a monad, the diagrams for the associative laws and unitary laws are shown in Figure 13. These diagrams provide the formal basis for the ap-

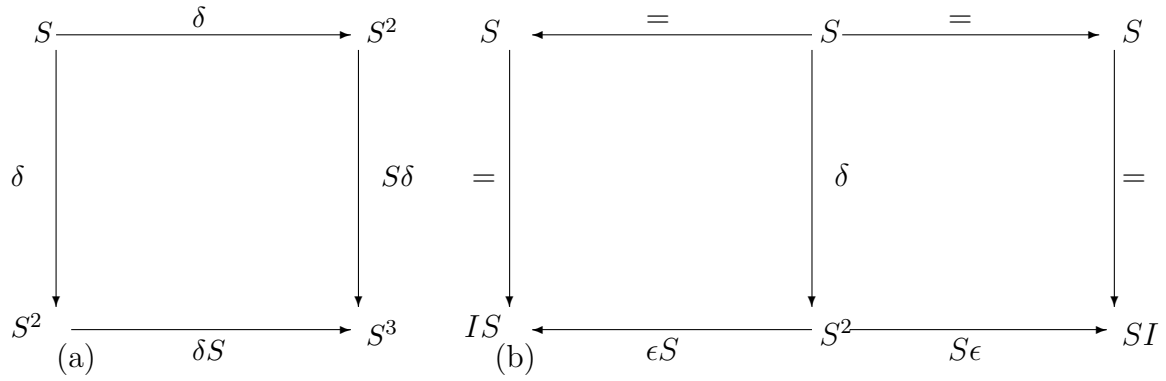

Figure 14: (a) Associative Law for Comonad $< S, \epsilon, \delta >$ (b) Left and Right Unitary Laws for Comonad $< S, \epsilon, \delta >$

proach. Figure 13(a) shows the relationship between T^3, T^2 and T where T is the endofunctor $GF : \mathbf{X} \longrightarrow \mathbf{X}$, $\mathbf{X}$ being any category. An endofunctor is a functor with the same source and target. A pair of adjoint functors F and G is an endofunctor as the source of $F : \mathbf{X} \longrightarrow \mathbf{Y}$ is $\mathbf{X}$ and the target of $G : \mathbf{Y} \longrightarrow \mathbf{X}$ is also $\mathbf{X}$. The unit or identity of the monad is $\eta : 1 \longrightarrow T$ from Figure 13(b) and the multiplication of the monad is $\mu : T^2 \longrightarrow T$ from Figure 13(a). We therefore write the monad T as the object $< T, \eta, \mu >$, with the category $\mathbf{X}$, on which the monad is based, omitted as it is inferred from the functors involved. However, it is not wrong to write the monad as the object $< \mathbf{X}, T, \eta, \mu >$ where the nature of $\mathbf{X}$ has a bearing on the arguments being made. Further it is often useful to say on which category the monad is based.

For a comonad, the dual of the monad, the diagrams for the associative laws and unitary laws are shown in Figure 14. Figure 14(a) shows the relationship between S, S^2 and S^3 where S is the endofunctor $FG : \mathbf{Y} \longrightarrow \mathbf{Y}$, $\mathbf{Y}$ being any category. The counit or identity of the comonad is $\epsilon : S \longrightarrow 1$ from Figure 14(a) and the comultiplication of the comonad is $\delta : S \longrightarrow S^2$ from Figure 14(b). We therefore write the comonad S as the object $< S, \epsilon, \delta >$ or $< \mathbf{Y}, S, \epsilon, \delta >$.

Figure 15 shows the two triangular identities for the monad in the category $\mathbf{X}$, derived by applying the interchange law to Figure 13(b). Through commutativity Figure 15 defines the arrow $G\epsilon F : GFGF \longrightarrow GF$. This

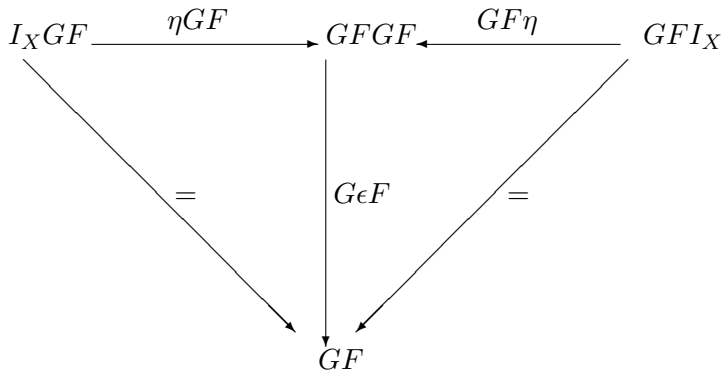

Figure 15: The Monad in the category **X**: Triangular Identities defining ϵ

arrow is the multiplication of Figure 13, that is $\mu : T^2 \longrightarrow T$. Therefore we can rewrite the monad $< T, \eta, \mu >$ as $< T, \eta, G\epsilon F >$ for an alternative view, based on the units and counits of adjunction, η and ϵ respectively.

5.5 Historical and Present Usage of the Monad Term

According to Hippolytus ($170 - 235$ AD), the worldview was inspired by the Pythagoreans, who called the first thing that came into existence the *monad*, from which came the dyad, triad, tetrad, etc. (Bunsen et al. 1854). Gnosticism is a modern term for a multitude of Jewish religious ideas and systems from the first and second century AD, with the highest God, Supreme Being or the One, termed the Monad. The Syrian-Egyptian school depicts creation as coming from a primal monadic source, finally resulting in the creation of the material universe.

The monad entered metaphysics as the *Monadology* of Leibniz, written from 1712-1714 as *Principes de la nature et de la grâce fondé en raison*, which has since been published in various forms and languages (Leibniz, 1714). Leibniz allows just one type of element in the building of the universe, which is given the name monad or entelechy, and described as a simple substance, which has no parts, hence indivisible. Monads are elementary particles with blurred perceptions of one another and have been described as eternal, indecomposable, individual, subject to their own laws, un-interacting, each reflecting the entire universe in a pre-established harmony; monads are centres of force; substance is force, while space, matter, and motion are merely phenomenal. Like atoms, monads are irreducible but differ in their complete

mutual independence, and in their following of a preprogrammed set of instructions peculiar to itself, so that a monad 'knows' what to do at each moment. Each monad is like a little mirror of the universe.

The monad term is also used in music, where it is a single note, with a dyad being 2 notes, a triad 3 notes, etc., and in biology where it is a unicellular organism.

In functional programming, the monad is an increasingly popular construction as an abstract data type, with promising developments in the language Haskell (Haskell, 2017; Lipovača, 2011), named after Haskell B Curry, who developed the transformation of functions through currying in the $\lambda-$calculus. The monad in Haskell is formally classified as an extension of the monad developed in category theory, involving the notion of a strong monad (Moggi, 1989; Mulry, 2013). Such a monad is defined in higher-order category theory as a bicategory construction. In more concrete terms a strong monad is defined as a (categorial) monad with strengthening with respect to products and idempotency. The strengthening with products leads to the concept of a Cartesian monad where, if the underlying categories are pullbacks, the monad T preserves pullbacks and μ and η are Cartesian, then the monad is Cartesian. Such a construction facilitates the use of T in transformations where a Cartesian type is expected. The strengthening with idempotency provides resilience as further operations are performed. So with the underlying category for the monad $\mathbf{X}$ being Cartesian with the object $A \times B$, there is a natural transformation $\tau_{A,B}$ from the Cartesian operation $(A \times TB)$ to $T(A \times B)$ such that strengthening with the identity I is immaterial, consecutive applications of strength commute, and strength commutes with monad unit and multiplication (Moggi, 1991). Further details of the Cartesian monad are found later in this paper in Section 7, in the work by Mulry (Mulry, 2013) and in Appendix C of Leinster's book *Higher Operads, Higher Categories* (Leinster, 2004).

Category theory is regarded as a unifying force so might be able to provide an insight into all of the above notions of the monad. The notion of unit applies to all the various usages and this is continued into the categorial version with the unit in the monad definition $< T, \eta, \mu >$ of $\eta : 1 \longrightarrow T$ and the counit in the comonad definition $< S, \epsilon, \delta >$ of $\epsilon : S \longrightarrow 1$. The monad of Leibniz is similar to the categorial version in respect of their following a preprogrammed set of instructions with each monad being a little mirror of the universe. However, there is a major difference – Leibniz's monad is a particle and the categorial monad is a process – emphasising the set-based

nature of Leibniz's work. The use of the term monad in music appears to reflect the physical reality of a single note. From a more constructive point of view, musical units, and hence monads, might also include chords and other logical combinations of notes. An application of the categorial monad to music is under active consideration. The use of the term monad for a unicellular organism has lapsed, maybe because the general term was confusable with its use for specific unicellular organisms, the *Monas*. The comparison between the monad of functional programming and that in category theory is the most useful: this shows that the Cartesian monad selected for functional programming is indeed the type of monad needed for information systems as the underlying Haskell category has products, in particular pullbacks, which form the basis of our structural approach.

6 Process on a Topos

The monad and comonad processes are applied to a topos, defining the structure of the data, to perform the transactions. The design of the processes ia therefore termed Monadic Design. We write the process on a topos as:

$$T : \mathbf{E} \longrightarrow \mathbf{E}'$$

where T is the Cartesian monad $< T, \eta, \mu >$ for a category $\mathbf{E}$ with endofunctor T, that is $GF : \mathbf{E} \longrightarrow \mathbf{E}$, unit of adjunction $\eta : 1 \longrightarrow T$ and unit of multiplication $\mu : T^2 \longrightarrow T$.

The source topos is $\mathbf{E}$ and the target topos is $\mathbf{E}'$, with the topos based on pullbacks, including the pasted types, as described in Section 3. The type (intension) of the source and target is the same but the data values (extension) will vary. Closure is achieved as the type is preserved.

For the running bank example, the Cartesian monad T is the banking system transaction, the source information system is $\mathbf{E}$ and the target information system is $\mathbf{E}'$. There may be more than one adjunction for a monad T, based on a category $\mathbf{E}$. For instance $< F, G, \eta, \epsilon >$ may be one adjunction for $\mathbf{E} \longrightarrow \mathbf{E}'$ with another of $< F_A, G_A, \eta_A, \epsilon_A >$ for $\mathbf{A} \longrightarrow \mathbf{E}$, where $\mathbf{A}$ is a subcategory of $\mathbf{E}$. So a variety of adjunctions may be handled by a single monad, over various subcategories of a particular category. This gives flexibility in handling different data-sets with the same underlying structure.

For the process there will also be a comonad:

$$S : \mathbf{E}' \longrightarrow \mathbf{E}$$

where S is the Cartesian comonad $< S, \epsilon, \delta >$ for a category $\mathbf{E}'$ with endofunctor S, that is $FG : \mathbf{E}' \longrightarrow \mathbf{E}'$, counit of adjunction $\epsilon : S \longrightarrow 1$, counit of multiplication $\delta : S \longrightarrow S^2$.

Categories of algebras can be defined over the monad and comonad. From the algebraic perspective, there are two approaches employing the monad/comonad as the underlying categories. The category of algebras over a monad is traditionally called its Eilenberg-Moore category (Eilenberg & Moore, 1965) ((Mac Lane, 1998) at pp. 139-142). Dually, the Eilenberg-Moore category of a comonad is its category of coalgebras. The subcategory of free algebras is traditionally called the Kleisli category of the monad, as is its dual the subcategory of co-free co-algebras of the comonad ((Mac Lane, 1998) at pp. 147-148). The Kleisli category of a monad transforms a monad into a form more suitable for implementation in a functional language such as Haskell. Compared to the EML form of Mac Lane, Kleisli strength generalises the notion of commutativity and guarantees that products lift to the corresponding Kleisli categories (Mulry, 2013). From the point of view of products, the monads developed to Kleisli strength are applicable in a much wider range of computing applications. Kleisli categories are discussed in more detail in Section 7.

6.1 The T-algebra

The T-algebras are one of the algebraic forms resulting from the work of Eilenberg and Moore (Eilenberg & Moore, 1965). Such algebras facilitate changing the definition of a monad and therefore permitting fundamental changes to the operand of our process. For any category $\mathbf{X}$, which in our case is a topos $\mathbf{E}$, the T-algebra produces a new consistent state of adjunction for a modified intension.

In more detail, applying the T-algebra to a topos $\mathbf{E}$, in the monad with adjunction $< GF, \eta, \mu >$, yields a new monad adjunction $< G^T F^T, \eta^T, G^T \epsilon^T F^T >$: $\mathbf{E} \longrightarrow \mathbf{E^T}$; that is a new monad adjunction $F^T \dashv G^T$ is defined to accommodate the changed category $\mathbf{E^T}$. A T-algebra is $< e, h >$ where e is an object in $\mathbf{E}$. The structure map of the algebra is $h : Te \longrightarrow e$ such that the diagrams in Figure 16 commute. Beck's Theorem provides rules on which categorial transformations in the T-algebra $X \longrightarrow X^T$ are valid (Beck, 2003). This is

sometimes called PTT (Precise Tripleability Theorem).

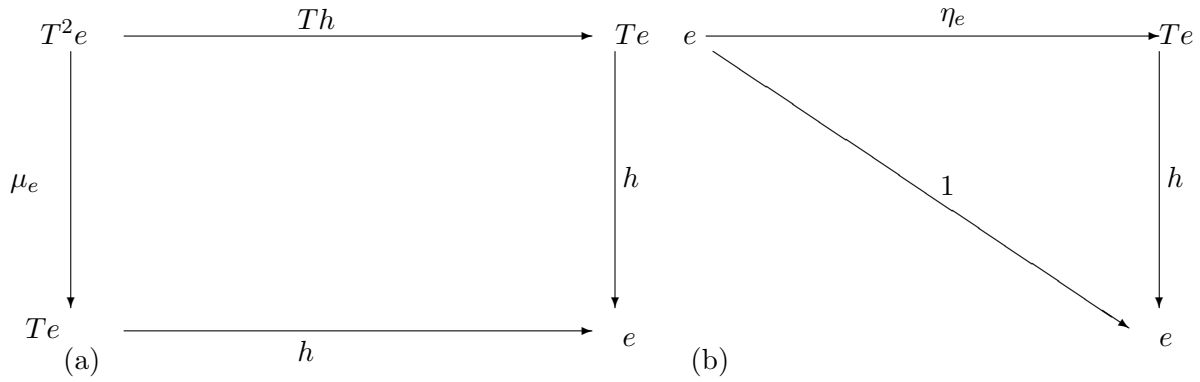

Figure 16: T-algebra: (a) Associative Laws, (b) Unitary Laws

7 Application

The categorial monadic approach is being used for the Blockchain (Meredith, 2015), a transaction system, adopted by Bitcoin, for keeping hundreds or even thousands of copies of each transaction record, using multiple transaction logs. The monadic design pattern provides a broad range of transactional semantics with composition the key to scaling any system. The blockchain approach is drawing interest from the established banking industry, where a blockchain is viewed as a shared, encrypted 'ledger' that cannot be manipulated, offering promise for secure transactions (Phys Org, 2015). Meredith indicates that compositionality is the key to reliability but offers few details on how this is achieved in the monad. Compositionality is a cornerstone of category theory, defined as a minimum up to some level of isomorphism. In monad/comonad definitions there is the choice of the Mac Lane (EML) or Kleisli algebras as introduced above in Section 6. It is the approach owing to Heinrich Kleisli that has elevated compositionality to a higher level, through the Kleisli lift, described for instance by Mulry (Mulry, 2013). In the diagram in Figure 17, H is a monad $< H, \eta, \mu >$ in $\mathbf{X}$ and K is a monad $< K, \gamma, \rho >$ in $\mathbf{Y}$. The Kleisli categories, representing the free algebras, are $\mathbf{X_H}$ and $\mathbf{Y_K}$. The Kleisli lift of functor F is the functor $\bar{F} : \mathbf{X_H} \longrightarrow \mathbf{Y_K}$ such that the diagram in Figure 17 commutes. Associated with this diagram is

the definition of the lifting natural transformation $\lambda : FH \longrightarrow KF$ in Figure 18, derived through applying the interchange law to the component functors and natural transformations in the two monads defined above.

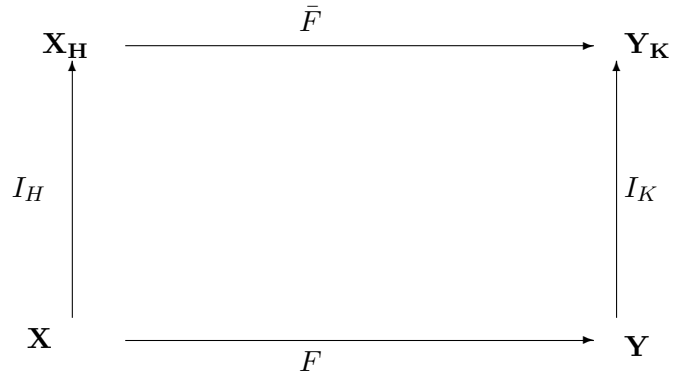

Figure 17: Kleisli Lifting of Functor $F : \mathbf{X} \longrightarrow \mathbf{Y}$ to $\bar{F} : \mathbf{X_H} \longrightarrow \mathbf{Y_K}$

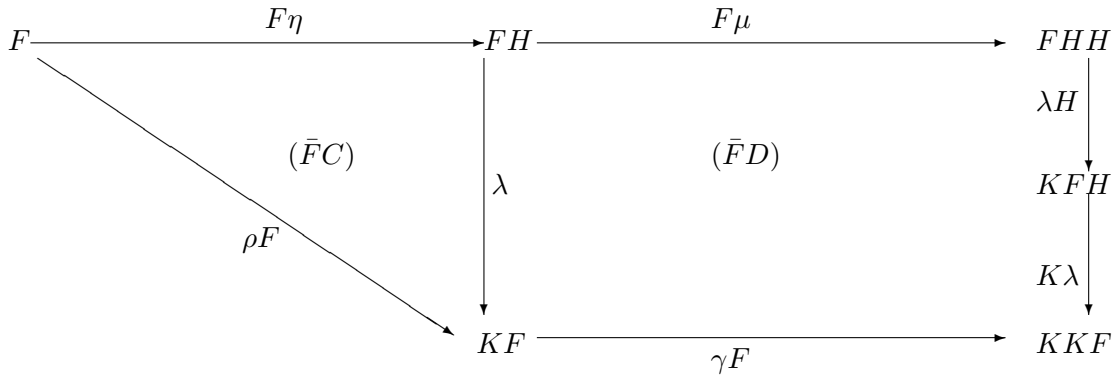

Figure 18: Kleisli Lifting of Functor F to $\bar{F}$: the lifting natural transformation $\lambda : FH \longrightarrow KF$

So far the Kleisli lift applies to any category, giving what is termed Kleisli prestrength. We now need to consider the Kleisli lifting of a bicategory, one involving a product of two categories. This is essential if the products are to be well defined for compositional purposes as indicated in Section 6. The lifting gives rise to what is termed Kleisli strength, forming the basis of the Cartesian monad, a term introduced earlier in our overview of the

Haskell programming language in Section 5.5. The terms Cartesian monad and strong monad encountered in the literature are for our purposes interchangeable. The enhanced compositionality is achieved firstly by defining a natural transformation $\tau_{A,B} : A \times TB \longrightarrow T(A \times B)$ for objects A, B, C in the category $\mathbf{X}$ with monad $< T, \eta, \mu >$ such that the diagram in Figure 19 commutes. A further natural transformation $\lambda_{TA} : I \times TA \longrightarrow TA$ is also defined, as shown in the commuting diagram in Figure 20, to reinforce the interchange laws employed in Figure 18. Both the diagrams defining the Cartesian monad involve the Cartesian product, the most relevant for information systems, but the theory is actually more general covering the tensorial (outer) product $A \otimes B$, which may have more relevance for studies involving vectors. Further diagrams are required when the product is tensorial, rather than Cartesian, involving multiplication through the arrow $\mu_{A \times B}$ and associativity though the arrow $\tau_{A,B \times C}$. A major advantage of Kleisli strength monads is that they can, in general, be composed naturally, unlike monads of weaker strength. Such composability increases reliability and scalability, both of which are vital for large scale information systems. Kleisli strength facilitates the discovery of distributive laws.

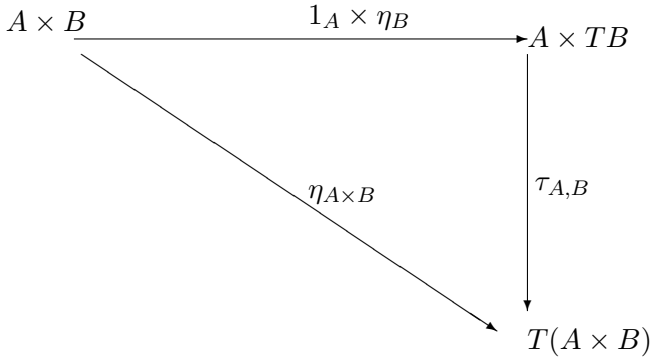

Figure 19: Cartesian Monad: Diagram defining the natural transformation $\tau_{A,B}$

Meredith (Meredith, 2015) envisages that the monadic design patterns, providing a broad range of transactional semantics, would have a front-end data sublanguage of the applied $\pi-$calculus, a compositional process calculus developed for concurrent programming by Milner (Milner, 1999). However, other presentational techniques from category theory are available, such as

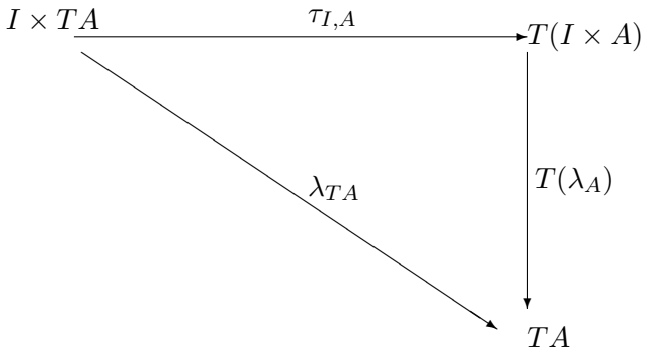

Figure 20: Cartesian Monad: Diagram defining the natural transformation λ_{TA}

bigraphs, and should also be evaluated before a choice is made.

In the functional programming language Haskell, monadic design patterns are employed. The design pattern for a category $\mathbf{C}$ is $\mathbf{H} =< H, \eta, \mu >$ where $\mathbf{H}$ is the monad with type constructor H, η is a return function, $\mu : HHA \longrightarrow A$ is a join function. In more conventional monad terminology H is the endofunctor, η the unit of adjunction and μ the multiplication (Mulry, 2013). If the monad is of the *Maybe* type, there are facilities for exception handling. To facilitate monad composition, the monad is lifted into a Kleisli category, with the power of a strong monad or a Cartesian monad. A monad composition operator, also known as the Kleisli composition operator, is available for composing one monad with another naturally (Diehl, 2013).

Returning to our banking example we can see that composition of processes is readily available if our monads are Cartesian, with the Kleisli lift. So for two monads $\mathbf{T} =< T, \eta, \mu >$ and $\mathbf{U} =< U, \gamma, \rho >$, we can write UT for the composite process, where say T is the banking transaction with checks for its feasibility and U is a task establishing remote mirror facilities, as in distributed data recovery systems, for recording the results persistently. Such compositionality could be enforced over large distributed systems by involving many individual monads. So monads can be used either in the small individually in a local environment or, through composition, in the whole universe of the information system. The efficacy of the monad approach can be proven through category theory, thereby increasing the reliability and robustness of a system, where every transaction is critical. Further the monad

can be directly implemented in the programming language Haskell, enabling experimental results to be derived. An approach has therefore been developed that is consistent with the process-relational philosophy, guaranteeing integrity in an experimental environment, with the monad as process and topos as relation.

8 Summary

The combination of the topos, as the underlying data-type, and the monad, as the process or transformer, appears to satisfy the requirements of information systems. The topos is based on pullbacks, which can be nested recursively or pasted together for complex relationships. The bottom level of Dolittle diagrams holds both the intension and extension for the data held. Data normalisation arises naturally through the rules of pullback construction. The subobject classifier of a topos facilitates internal queries on the information system. The monad is defined as three components for operations on a category: an endofunctor that is often an adjunction, the unit of adjunction and the unit of multiplication. There are two main approaches for applying the monad as an algebra: Eilenberg-Moore (EML) and Kleisli. The Kleisli approach finds favour, with its lift to Cartesian monads handling products, providing compositionality across a succession of monads and a route for experimental implementation in Haskell. The categorial features employed of monad and topos correspond to process and relation respectively in the process-relational philosophy.

9 Acknowledgements

We am very grateful to Michael Brockway of Department of Computer Science and Digital Technologies, Northumbria University, for helpful discussions on the theory of pasted pullbacks and monads.

References

Adámek, Jiří, Herrlich, Horst, Strecker, George E. (2005). Abstract and concrete categories, John Wiley (1990). Recent edition at `http://katmat.math.uni-bremen.de/acc`.

Banach, R. (1994). Regular relations and Bicartesian Squares, Theoretical Computer Science **129**(1) 187-192. https://doi.org/10.1016/0304-3975(94)90086-8

Beck, Jonathan Mock. (2003). Triples, Algebras and Cohomology, Reprints in Theory and Applications of Categories, Columbia University PhD thesis, 2: 159, MR 1987896, originally published 1967. http://www.tac.mta.ca/tac/reprints/articles/2/tr2abs.html

Bunsen, Christian Karl Josias, Freiherr von; Hare, Julius Charles & Bernays, Jacob. (1854). Hippolytus and his Age, published Longman, Brown, Green, & Longmans, London 577 pp.

Diehl, Stephen. (2013). Monads made difficult. http://www.stephendiehl.com/posts/monads.html

Eilenberg, Samuel, & Moore, John C. (1965). Adjoint functors and triples, Illinois J Math **9**(3) 381-398. http://projecteuclid.org/euclid.ijm/1256068141.

Follett, M P. (1919). Community is a process, Philosophical Review **28** 576588. http://economics.arawakcity.org/node/95

Freyd, Peter, & Scedrov, Andre. (1990). Categories, Allegories. Mathematical Library **39** North-Holland.

$\lambda-$Haskell: an advanced, purely functional programming language. (2017). https://www.haskell.org/

Heather, Michael, & Rossiter, Nick. (2005). Logical Monism: The Global Identity of Applicable Logic, Advanced Studies in Mathematics and Logic **2** 39-52. http://nickrossiter.org.uk/process/advstudiesmathsmonism.pdf

Heather, Michael, & Rossiter, Nick. (2011). The Process Category of Reality, ANPA 31, Cambridge 224-262. http://nickrossiter.org.uk/process/anpa0911.pdf

Johnson, M & Rosebrugh, R. (2002). Sketch data models, relational schema and data specifications. Electron Notes Theor Comput Sci **61** 51-63. http://www.mta.ca/~rrosebru/articles/sdmrsds.pdf

Kent, William. (1983). A Simple Guide to Five Normal Forms in Relational Database Theory, Communications of the ACM 26(2) 120-125. http://www.bkent.net/Doc/simple5.htm

Lambek, J, & Scott, P J. (1986). Introduction to Higher Order Categorical Logic, Cambridge. https://github.com/Mzk-Levi/texts/blob/master/LambekJ.,ScottP.J.IntroductiontoHigherOrderCategoricalLogic.pdf

Lawvere, F W, (1969), Adjointness in Foundations, Dialectica **23** 281-296.

Leinster, Tom. (2004). Higher Operads, Higher Categories, London Mathematical Society Lecture Note Series 298, Cambridge.

Leibniz G W. (1714). Monadologie; translated by Nicholas Rescher, 1991. The Monadology: An Edition for Students. University of Pittsburgh Press. Ariew and Garber 213, Loemker 67, Wiener III.13, Woolhouse and Francks 19. Online translations: Jonathan Bennett's translation; Latta's translation; French, Latin and Spanish edition, with facsimile of Leibniz's manuscript at the Wayback Machine (archived July 4, 2012); further editions établie par E Boutroux, Paris LGF 1991; Lamarra, A, Contexte GènGètique et Première Réception de la Monadologie, Revue de Synthese 128 311-323 (2007).

Levene, Mark, & Vincent, Millist W. (2000). Justification for inclusion dependency normal form, IEEE Transactions on Knowledge and Data Engineering **12**(2), pp. 281-291. http://eprints.bbk.ac.uk/196/1/Binder1.pdf

Lipovača, Miran. (2011). Learn You a Haskell for Great Good!, A Beginner's Guide, William Pollock, San Francisco.

Mac Lane, Saunders. (1998). Categories for the Working Mathematician, 2nd ed, Springer.

Meredith, Lucius Greg. (2015). Monadic design patterns for the Blockchain, DEVCON1, Ethereum Developer Conference, Gibson Hall, London, 9-13 Nov. https://www.youtube.com/watch?v=uzahKc_ukfM&feature=youtu.be

Mesle, C Robert. (1993). Process Theology: A Basic Introduction.

Milner, Robin. (1999). Communicating and Mobile Systems: The $\pi-$calculus, Cambridge.

Moggi, Eugenio. (1989). Computational Lambda-Calculus and Monads, Proceedings of the Fourth Annual Symposium on Logic in Computer Science 1423.

Moggi, Eugenio, (1991). Notions Of Computation And Monads, Information And Computation **93** 5592.

Mulry, Philip. (2013). Notions of Monad Strength, Banerjee, A, Danvy, O, Doh, K-G, Hatcliff, J, (edd.) David A. Schmidts 60th Birthday Festschrift, EPTCS 129, 6783, doi:10.4204/EPTCS.129.6. https://arxiv.org/pdf/1309.5132.pdf

ncatlab. (2018). Pullback as an Equalizer. https://ncatlab.org/nlab/show/pullback

Phys Org. (2015). Bitcoin's 'blockchain' tech may transform banking. http://phys.org/news/2015-12-bitcoin-blockchain-tech-banking.html

Rossiter, B N, Heather, M A, & Sisiaridis, D. (2006). Process as a World Transaction, Proceedings ANPA 27 Conceptions, 122-157. http://nickrossiter.org.uk/process/anpa064.pdf

Rossiter, Nick, & Heather, Michael. (2015). Formal Natural Philosophy: Top-down Design for Information & Communication Technologies with Category Theory, ANPA 35, Explorations, Grenville J Croll, Nicky Graves Gregory (edd.), 155-193. http://nickrossiter.org.uk/process/anpa-2015-a5-Latex.pdf

Rossiter, Nick, & Heather, Michael. (2016). Abstract Relations and Allegorical Categories, ANPA 36, Explorations II, Anton L. Vrba (ed.) 103-134. http://nickrossiter.org.uk/process/Rossiter-ANPA-PROC-36updated.pdf

Stout, Margaret, & Staton, Carrie M. (2011). The Ontology of Process Philosophy in Follett's Administrative Theory, Administrative Theory & Praxis, **33**(2) 268-292. http://www.tandfonline.com/doi/abs/10.2753/ATP1084-1806330206?journalCode=madt20

Stout, Margaret, & Love, Jeannine M. (2015). Relational Process Ontology, A Grounding for Global Governance, Administration & Society **47**(4) 447-481. http://journals.sagepub.com/doi/pdf/10.1177/0095399713490692

Whitehead, Alfred North, & Russell, Bertrand. (1910). *Principia Mathematica* **1** Cambridge University Press.

Whitehead, Alfred North. (1929). Process and Reality: An Essay in Cosmology.. Macmillan, New York; corr.ed., eds. David Ray Griffin and Donald W. Sherburne, New York: Free Press (1978). https://monoskop.org/images/4/40/Whitehead_Alfred_North_Process_and_Reality_corr_ed_1978.pdf