Mathematical Support for Ensemble Engineering*

Michael Johnson

Macquarie University, Sydney, Australia
mike@ics.mq.edu.au

Abstract. We study some of the mathematical challenges presented by the need to support ensemble engineering, concentrating on likely contributions from category theory and universal algebra. Particular attention is paid to dealing with missing data, modelling dynamics and interaction, and analysing inconsistencies.

Keywords: Ensemble engineering, category theory, universal algebra, inconsistency analysis.

1 Introduction

This volume, and much of the earlier work of the INTERLINK Working Group 1 (WG1), has advocated *ensemble engineering* as an important new computing paradigm. For a discussion of *ensembles*, the reader is referred to [3] in which ensemble engineering is defined (page 19) as "the science and engineering discipline of complex, integrated ensembles of computing elements ... [and] ways to reliably and predictably model, design, and program them".

The growth in the development of distributed systems, mobile technologies, agent-based systems, multi-processor embedded systems, and the construction of interoperations for legacy systems all contribute engineering techniques that can be useful for ensemble engineering. At the same time, the challenges of very large numbers of nodes, adaptive technologies which blur the boundary between development time and operation time, open environments, and emergent behaviour, will require the development of new mathematical techniques for reliable design.

Some of the mathematics expected to be of value can be predicted already. Probabilistic analyses, differential equations, the modal logic of games, and many aspects of complex systems theory are all relevant. This paper explores the prospects of providing mathematical support for ensemble engineering using one less known mathematical tool — category theoretic universal algebra. After some background material and a brief example of specification via category theoretic universal algebra we deal in turn with the need to develop mathematics for the

* Research partially supported by the Australian Research Council. I thank the anonymous referees for useful comments and the editors for encouraging me to include the survey material in Sections 2 and 3.

dynamics of algebras, proposals for the study and management of the integration of systems, how to deal with limited data at the local level of computational elements, how to deal with limited data during interoperation between computational elements, and a proposal for the analysis of systems in the presence of inconsistencies.

2 Remarks on Category Theory

Some readers of this volume will have little or no experience with the branch of mathematics known as category theory. The remainder of this paper attempts to outline results which follow from category theoretic analysis without including the mathematical details. Of course, there is a risk of "falling between two stools" — those who understand category theory may feel cheated by the missing details, while those with no experience of category theory may wonder what it's all about.

To try to address the first group, the category theorists, I have included precise statements of the category theory involved whenever I can do that with a few words. Those without experience of category theory should just read past the technical terms when they appear.

For the second group I say a few words informally about category theory in this section. Mostly I try to avoid full definitions, talking *about* category theory rather than trying to provide an exposition of a graduate course in a page or two.

Nevertheless, we begin with some detailed definitions: A *category* is a directed (multi-) graph, together with a composition for arrows in the graph defined whenever two arrows meet head to tail (viz $A \longrightarrow B \longrightarrow C$, with the resulting composite a single arrow from A to C). If the two composable arrows are called f and g then the composite is denoted gf (note the order which corresponds to the usual (algebraic) notation for composite functions). The composition operation is required to be associative (so $h(gf) = (hg)f$) and to have identities.

Since in every category composition of arrows is associative, any string of arrows $A \longrightarrow B \longrightarrow \ldots \longrightarrow T$ in a category has a unique composite. Of course, different strings might have the same composite. If another string of arrows $A \longrightarrow C \longrightarrow \ldots \longrightarrow T$ has the same composite the diagram made up of the two strings is said to *commute*. Some examples of commutative triangles and a commutative square appear in Section 4.

Common examples of categories arise from classes of mathematical structures and the arrows, often called *morphisms*, between them. For example, the category of finite sets has as objects all finite sets and as arrows the functions between the sets. Similarly there are categories whose objects are groups and whose arrows are group homomorphisms; topological spaces with continuous maps; graphs with graph homomorphisms; etc. There is even a category whose objects are all "small" categories and whose arrows are the appropriate morphisms for categories — graph morphisms ϕ which respect the composition meaning that $\phi(gf) = \phi(g)\phi(f)$ and ϕ takes identity arrows to identity arrows — called *functors*. (Do not be concerned about "size" issues which cause no difficulties provided one correctly uses classes).

Finite categories are quite common in computer science applications. They are easily presented by giving their finite underlying graph and tabulating the (finitely many) compositions.

Categories were introduced during the 1940s, originally to make precise the notion of *natural transformation* — a kind of morphism between functors. The language of categories has since become widely used in many ares of mathematics, and category theory itself has had an important role in unifying and organising disparate areas of mathematics.

Categories are disarmingly simple — a category is merely a graph together with a composition which is associative and has identities. What is surprising is that such a simple notion can have much "semantic power". A large part of that power comes from the discovery during the 1950s that notions nowadays known as *limits* and *colimits*, including *products*, *coproducts*, *pullbacks* and *pushouts*, can be described solely as properties of arrows within a category. Such properties typically take the form "for all arrows of a certain kind there is a unique arrow of another kind making certain diagrams commute", and because of the initial universal quantifier ("for all") they are known as *universal properties.* Another kind of universal property, *cartesian morphisms*, plays a role in Section 8.

The existence of objects with certain universal properties is sometimes called an *exactness condition.* For example, to say that a category, like the category of finite sets, has all finite products (which it does), is an exactness condition. A category which has all finite limits is called *finitely complete.* Finite completeness is quite a strong exactness condition, but we do sometimes require more, for example the existence of finite coproducts (Section 4). The category of finite sets has all of these exactness conditions and more besides.

3 Categorical Universal Algebra

In the 1960s, F.W. Lawvere discovered and developed categorical universal algebra. Lawvere observed that using finite products and commutative diagrams he could construct a category which encapsulated all of the information required for a particular branch of algebra, say group theory. The category is called the *theory* of a group. Every finite group arises as a finite product preserving functor from the theory to the category of finite sets. Indeed more: The category whose objects are such functors and whose arrows are the natural transformations between them is *equivalent* to the category of finite groups and group homomorphisms.

Similarly other areas of algebra arise correspondingly from other theories. Monoids, semigroups, rings and many other algebraic structures can be treated in exactly the same way. And theorems proved about the categories using category theoretic tools are theorems of universal algebra.

In order to further generalise from fully defined operations (like the product of elements of a group) to partially defined operations (like the composition of arrows in a category) one needs to replace "finite products" with more general limits. For example, a certain pullback can be used to abstractly specify the *composable pairs* of arrows in an abstract category. Thus, one is led to the notion of a *theory* as a category with certain exactness properties, and algebras as functors from the theory to a category of sets which preserve the exactness properties. Commutative diagrams in the theory still correspond to axioms that the algebras are required to satisfy.

In the 1970s Lawvere and others developed categorical logic, most explicitly in *topos theory*, in which certain universal properties can be used to represent standard logical constructions. Categorical logic is important to us here because category theory can be used to specify systems as well as algebras, and restrictions on those systems are often expressed in logical terms which can, since the 1970s, be rephrased into category theoretic specifications.

To conclude this very brief historical survey we note that in the 1980s people began widely using category theory and universal algebra (although not usually category theoretic universal algebra) for specifications in computer science. In the 1990s the author and others used Lawvere style category theoretic universal algebra to specify information systems while others used category theory for program language semantics or even to introduce programming constructs (eg *monads* in Haskell). Since 2000 the author and his colleague Rosebrugh have been using category theory to study view updates (Section 8), and view updates to engineer system interoperations (for arguably very very small ensembles).

4 System Specification Using Universal Algebra

This section briefly reviews the mathematical foundation the author has used for system specification using category theory. It is based on categorical universal algebra, which is the basis of classic algebraic specification techniques [2]. We assume some familiarity with elementary category theory, as might be obtained in [1], [11] or [13]. For the purposes of this paper we will just outline the basic ideas. A fuller treatment can be found in, for example, [7].

A *theory* is a finitely complete category, frequently with other exactness properties (for example finite coproducts in much of the author's work with his colleagues Rosebrugh and Dampney). A *specification* is a presentation for a theory, usually given via a *sketch* [1]. A *model* or *state* for a theory is a finite limit preserving, and whatever other exactness properties might have been specified preserving, functor from the theory to the category of finite sets. A model is also called in more mathematical treatments an *algebra*.

A model should be thought of as a snapshot of the system in operation, while the theory constrains the possible snapshots — in a sense the theory embodies all of the information that is required in all possible snapshots.

These formalities allow us to be quite precise about our systems, and to begin to analyse them mathematically. but they also have other advantages some of which we will note here:

– The theory is invariant — there are usually many different presentations of the same system, and we don't wish to deal with artifacts of any particular presentation. For any given system the theories will be equivalent categories no matter which presentation is used.

– The theory includes the constraints, axioms or business rules that are important to capture at specification time, and to enforce at operation time. Indeed, the power of the exactness properties permits, via categorical logic, the specification and enforcement of logically delicate constraints.
– The theory can be constructed so as to include mathematical representations of the data and properties that can be derived from particular states (models) of the system. For example, when the system in question is a database, the theory will include (automatically because of the required finite limits) representations of all of the *queries* that can be applied to the database.

Example 1. Figure 1 is fragment of a theory for a health informatics system [5]. Models of this theory include sets representing for example all of the in-patient operations for which details are stored in the system and all of the hospitals for which details are stored in the system, along with a function between them indicating which operation took place in which hospital. The theory itself includes many more base datatypes together with nodes representing all possible queries of the database, and arrows representing all derived operations among datatypes.

In a little more detail:

– The graph shown is a type diagram. The three monic arrows (those with extra tails to indicate that they should be realised as injective functions) indicate subtypes. The other arrows are functions (operations) which given an instance of their domain type will return an instance of their codomain type. The names on nodes and arrows have no formal significance but do indicate the real world semantics being captured in the specification and will be used from now on in our discussion.
– The commutativity of the two triangles represents a typical real-world constraint: Every in-patient operation conducted at a particular hospital by a particular medical practitioner must take place under a practice agreement (a type of contract) between that hospital and that practitioner. If, instead

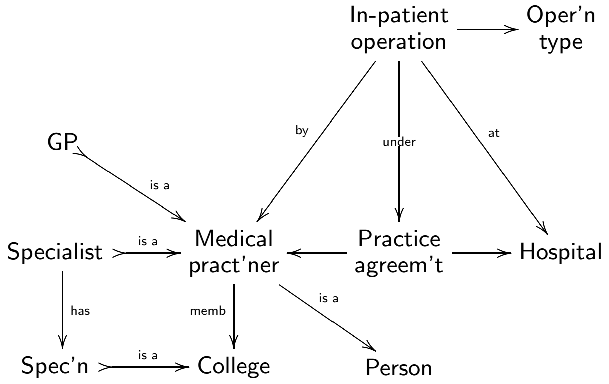

Fig. 1. A fragment of a theory for a health informatics system

the left hand triangle were not required to commute then it would still be the case that every operation took place under an agreement, but Dr A could operate under Dr B's practice agreement. In many information models, situations like this do not even include the arrow marked under, and thus they store the contractual information, but do not specify the constraint — it is expected to be added at implementation time.

- The square is also commutative and is required to be a pullback. This ensures that the specialists are precisely those medical practitioners who are members of a college which occurs in the subtype Specialisation. This is important because the registration procedures (not shown) for specialists are different from those for other medical practitioners.
- Subtype inclusion arrows, and other arrows that are required to be monic in models, are so specified using pullbacks.
- As is common practice, attributes are not shown in Figure 1, but they are important. They are usually large fixed value sets, often of type `integer` (with specified bounding values), `string` (of specified maximum length), `date` etc. Some examples for this theory include the validity period of a practice agreement, the name and the address of a person, the classification of a hospital, the date of an operation, the provider number of a medical practitioner and many more. Strictly, they are all part of the theory.

It is worth noting that in contrast with most algebras that arise in mathematics, information systems are usually very many-sorted (based on many different sets like Hospital, Person, College etc) and most operations are unary (at, by, isa etc). Also in many algebraic specifications essentially unique models (algebras) are sought, frequently by taking initial algebra semantics for example. In contrast ensembles frequently need to collect data maintaining histories or sets of instances for each datatype.

5 Dynamics

The utility of categorical universal algebra for abstract specification of software and systems, including areas as diverse as information systems and programming language semantics, is well-established. So, rather than rehearsing those arguments we will begin by considering one of the limitations of present work — the mathematical treatment of the *dynamics* of algebras is relatively undeveloped in modern universal algebra.

Universal algebra specifies and studies individual algebras, or varieties of algebras with certain properties, but only rarely does it deal with algebras changing through for example adding or deleting an element. Yet this dynamic nature is central for the study of systems that process information — a snapshot of the system at a moment in time is an algebra, and as the system acquires more information, perhaps by the addition of a new instance of some type (eg, an insertion of a new entity instance in a database), the algebra is modified to become another algebra.

One well-understood, but fairly limited, instance of algebra dynamics is the extension of a field or a ring by an indeterminate ($R \longrightarrow R[x]$). In a more general sense algebras can be modified by quotienting or by operating on them with other algebras (taking for example the product — see the next section). But there is little theory to support "Given a group G, add an element with the following properties (expressed in terms of other elements of G) to get a new group G'".

A certain amount of theoretical development of dynamics has been done in [8] which developed a mathematical foundation that unifies the treatment of specifications, updates (dynamics), and categories of algebras for a class of database systems. Nevertheless, much remains to be done.

6 Interactions among System Components

One of the outstanding features of ensembles of computational elements is the interaction of those elements. While ensemble interaction should be dynamic, and possibly adaptive, it is nevertheless important to design and manage interactions and to model them mathematically.

The traditional universal algebra approach to modelling and designing interactions between systems involved calculating pushouts in the category of theories. This has been an effective technique, but it may be seen as less appropriate for ensembles as it views the ensemble as a static system constructed from parts.

R.F.C Walters and his colleagues have an ongoing programme of research into a (bi-) categorical calculus of processes which accurately models the composition of systems including representations of concurrency and feedback (see for example [9] and [10]). The processes may be viewed as algebras in our framework, and algebras can be composed using algebra operations akin to product, sum, and trace. More recently the researchers have incorporated timing issues into their mathematical model. It seems likely that approaches such as these will be very useful in ensemble engineering, although much of the work is still oriented towards viewing the ensemble as a constructed system rather than as a dynamic evolving agglomeration.

Another approach advocated by the author [4] focuses on managing the communications between extant systems using techniques outlined in Section 8. The basis for studying interactions is quite like the pushout approach: We begin with a span of theories $\mathbb{E} \longleftarrow \mathbf{V} \longrightarrow \mathbb{E}'$, two of which, $\mathbb{E}$ and $\mathbb{E}'$, are the theories of independent computational elements while the third ($\mathbf{V}$) is a representation of their interactions, typically the common information on which they will attempt to remain synchronised. The synchronisation techniques (Section 8) are quite different from the calculation of a pushout, but more importantly for this section the systems specified by $\mathbb{E}$ and $\mathbb{E}'$ remain independent and can in principle move in and out of communication rather than being parts of a composite system calculated via pushout.

So, we have at least three promising techniques to analyse and design interactions among elements of ensembles. In all three cases there is still much to do to develop the mathematical approaches to more fully support ensemble engineering.

7 Limited Data at Computational Elements

One of the most significant aspects of an ensemble is the need to deal with semi-structured rather than fully-structured data. In a dynamic world with many adaptive elements joining and leaving an ensemble, it's hard to imagine how complete sets of data can be captured, communicated and maintained.

The problem of incomplete data has long been dealt with in the database community by using NULLs — invalid data values that act as place holders for missing data. Nevertheless, NULLs have had at best an ambiguous status in the theory, frequently they were retro-fitted long after a design which de facto assumed perfect data availability. And when they were considered in the theory they led to widely differing treatments using so-called three-valued logics.

The first observation to be made here is that missing values have no importance in and of themselves — one doesn't need a NULL value to store the fact that there is nothing to store. The delicacy in dealing with missing values arises because *operations* may need to take undefined values.

Now partial operations are easily represented category theoretically. Suppose $f : A \longrightarrow B$ is an operation which might be only partially defined on its domain A. Then in the theory which establishes the type of f we don't include an arrow $A \longrightarrow B$ but rather a span $A \longleftarrow A' \longrightarrow B$. Thus A' is the type standing for those instances of A on which f is defined, and notice that in a dynamic system this provides full flexibility — instances of elements of A can be added to, or deleted from, A' as information becomes available.

Interestingly, and delicately, the span approach is not equivalent to using NULLs. In the latter case a partially defined $f : A \longrightarrow B$ would be represented by a fully defined $f' : A \longrightarrow (B+1)$ where the extra element of the codomain is the NULL. The two approaches are Morita equivalent (ie have equivalent categories of models, see [6]) on the assumption that the subobject $A' \rightarrowtail A$ is complemented and this is not usually the case. The difference is important in dynamic environments: To define f on a new value $a \in A$ is *extra* information and is represented in the span case by inserting a into A', but when f' has codomain $B+1$ extending the domain of definition really means reassigning values for f' (formerly $f'(a) = 1$, but once f becomes defined at a then $f'(a) = b$ for some $b \in B$).

8 Limited Data during Interoperation

Another source of limited data arises in dynamic environments when, for example, systems interoperate via a span of theories $\mathbb{E} \longleftarrow \mathbf{V} \longrightarrow \mathbb{E}'$ as proposed in Section 6.

Suppose we aim to keep, as far as possible, the algebra for $\mathbb{E}$ synchronised with the algebra for $\mathbb{E}'$ on common parts indicated by $\mathbf{V}$. In categorical universal algebra an algebra for a theory $\mathbb{E}'$ is an appropriate functor $\mathbb{E}' \longrightarrow \mathbf{Set}$. Thus an algebra for $\mathbb{E}'$ yields an algebra for $\mathbf{V}$ by composition with the theory morphism $\mathbf{V} \longrightarrow \mathbb{E}'$. Now, how can we modify the algebra (system snapshot) for $\mathbb{E}$ so that it remains synchronised with the new algebra for $\mathbf{V}$? This is the *view update problem*. The problem is genuinely difficult because of missing data. A change in the algebra for $\mathbb{E}'$ by an insert say, may result in a change in the algebra for $\mathbf{V}$ also by an insert. That insert is properly defined, since all the information that a $\mathbf{V}$-algebra needs is available in the $\mathbb{E}'$-algebra. But trying to propagate the insert to the extant $\mathbb{E}$-algebra may be impossible (if for example there are constraints that are required to be satisfied by $\mathbb{E}$-algebras that do not appear in $\mathbf{V}$-algebras), or ambiguous (if for example there are operations in $\mathbb{E}$-algebras that are not fully determined by operations in $\mathbf{V}$-algebras).

To engineer effective interoperations we need to determine those occasions when view-updating may be impossible since they are real limitations to interoperations, and for those situations where view-updating may be ambiguous because there are multiple solutions we seek a "best" solution — one for which the change to the $\mathbb{E}$-algebra is minimal. In category theoretic terms we seek a universal solution to the view updating problem and analysing the situation shows that the solution is given by well-known cartesian and op-cartesian morphisms [7].

Importantly the two types of missing data (treated in this section and the preceding section) interact well: In work still being written up the author shows that when operations support missing data using the span approach outlined in the previous section, and an insert leads to an ambiguous view update because such an operation is not fully-determined, the least defined extension of the operation will be a component of an op-cartesian morphism.

Of course there is much work to be done in testing the utility of these approaches for full ensemble engineering, but they have already proved their value in smaller scale system interoperations.

9 Analysing Systems in the Presence of Inconsistencies

Finally we consider one important mathematical limitation that can arise in dealing with ensembles. With loosely coupled, or indeed uncoupled, dynamic systems of open computing elements one can't ensure consistency — unexpected or malfunctioning elements might occasionally join an ensemble and exhibit properties which contradict ensemble invariants. This shouldn't be surprising. Conflicting systems often exist, and frequently operate effectively for extended periods in at least a narrow domain in the real world. But for mathematical tools such inconsistencies can be damning. A single inconsistency in a mathematical model invalidates everything that the model purports to demonstrate.

In fact, it is easy to see how the difference arises. Real systems have various flows of control and inconsistencies can co-exist for extended periods without being invoked together and coming into conflict. The system behaviour is determined by the traces of execution. In contrast mathematical structures exist in their Platonic entirety. If a contradiction exists, its effects cannot be distinguished from deductions that would have been valid in its absence. Every "behaviour" of the mathematical structure, everything that it proves, is brought into doubt by the presence of the inconsistency.

Recent work by Catherine Menon [12] addresses this problem using ideas from view updating. Menon develops what she calls CCF, the categorical consistency framework.

Menon has developed a framework which maintains mathematically the distinctions between modules and permits an analysis of the modules and an exploration of their joint consistency without in fact building them all into a mathematical model which would itself exhibit any inconsistency which was present.

Menon's work is a first step in an area that will need to become well-developed if we are to provide proper mathematical support for truly open and dynamic ensembles rather than using older mathematical techniques to analyse snapshots of ensembles as large static systems constructed from fixed components.

10 Conclusion

It is clear that a great range of mathematical techniques will be important for ensemble engineering. Some exist. Others will be developed to meet the new challenges that arise.

This paper has focused particularly on categorical universal algebra and explored some of the probable applications, and some of the current limitations, of extant universal algebra for ensemble engineering. We've demonstrated that some of the difficult problems of ensemble engineering can be addressed using recent developments based on categorical universal algebra, particularly the representations of missing data and the solution of view update problems, neither of which played a part in earlier applications of universal algebra. We look forward to many more similar developments in mathematical support for ensemble engineering.

References

1. Barr, M., Wells, C.: Category theory for computing science, 2nd edn. Prentice-Hall, Englewood Cliffs (1995)
2. Ehrig, H., Mahr, B.: Fundamentals of algebraic specifications. Springer, Heidelberg (1985)
3. Hölzl, M., Wirsing, M.: State of the Art for the Engineering of Software-Intensive Systems. Deliverable Number D3.1 (2007) (accessed July 19, 2008), `http://interlink.ics.forth.gr/central.aspx?sId=84I238I744I323I344283`
4. Johnson, M.: Enterprise Software with Half-Duplex Interoperations. In: Doumeingts, G., Mueller, J., Morel, G., Vallespir, B. (eds.) Enterprise Interoperability: New Challenges and Approaches, pp. 521–530. Springer, Heidelberg (2007)
5. Johnson, M., Rosebrugh, R.: View Updatability Based on the Models of a Formal Specification. In: Oliveira, J.N., Zave, P. (eds.) FME 2001. LNCS, vol. 2021, p. 534. Springer, Heidelberg (2001)
6. Johnson, M., Rosebrugh, R.: Three approaches to partiality in the sketch data model. ENTCS 78, 1–18 (2003)

7. Johnson, M., Rosebrugh, R.: Fibrations and Universal View Updatability. Theoretical Computer Science 388, 109–129 (2007)
8. Johnson, M., Rosebrugh, R., Wood, R.J.: Entity-relationship models and sketches. Theory and Applications of Categories 10, 94–112 (2002)
9. Katis, P., Sabadini, N., Walters, R.F.C.: Bicategories of processes. J. Pure Appl. Algebra 115, 141–178 (1997)
10. Katis, P., Sabadini, N., Walters, R.F.C.: On the algebra of systems with feedback and boundary. Rendiconti del Circolo Matematico di Palermo Serie II Suppl. 63, 123–156 (2000)
11. Mac Lane, S.: Categories for the Working Mathematician, 2nd edn. Graduate Texts in Mathematics 5. Springer, Heidelberg (1998)
12. Menon, C.: A category theoretic approach to inconsistencies in modular system specification. PhD thesis, University of Adelaide (2006)
13. Walters, R.F.C.: Categories and Computer Science. Cambridge University Press, Cambridge (1993)