On Category Theory as a (meta) Ontology for Information Systems Research

Michael Johnson
Department of Computing, Macquarie University, Sydney, Australia
Email: mike@ics.mq.edu.au
and
C.N.G. Dampney
Department of Computing, Macquarie University, Sydney, Australia
Email: kit@ics.mq.edu.au

Abstract: Category theory is a branch of mathematics that is renowned for its semantic power despite its very simple axiom set. The use of category theory as a meta-ontology for abstract algebra via the highly developed field of universal algebra has suggested that it be used as a foundation for research in computing. To date it has been widely used in, inter alia, the definition of abstract data types, the semantics of programming languages, and the design of functional programming languages. This paper illustrates the use of category theory as a meta-ontology for information systems research. It is based on the authors' extensive consultancy work using category theory to solve real problems in industrial applications involving information systems. In addition to discussing the role of category theory as an ontological tool for information systems research, the paper illustrates its use with a number of examples including system specification, the definitions of views and view updates, and system interoperations.

Categories & Descriptors: H.2.1 [**Database Management**]: Logical design—*data models*; F.3.1 [**Logics and Meanings of Programs**]: Specifying and verifying and reasoning about programs—*specification techniques*; D.2.4 [**Software Engineering**]: Software verification—*formal methods*

General Terms: Design, Management, Standardization, Theory

Keywords: Category theory, data modeling, information systems, interoperating systems, ontology, view update

1. Introduction

Among the important goals of ontological foundations for information systems is the ability to provide a more detailed understanding of semantics for data models. The need for semantic data models was outlined by Myopoulos [19], who also lamented the paucity of such models. A deeper understanding of the ontological issues surrounding data modeling is necessary for true semantic data modeling.

A data model is essentially a mathematical structure which, like most mathemat-

---

FOIS '01, October 17-19, 2001, Ogunquit, Maine, USA.

ical structures, is intended to correspond in more or less detail to some domain of activity and information in the world. Without underestimating the very interesting problems to do with the ontology of the things in that domain, our research program concentrates on bringing to bear semantic understanding from mathematics to extend our understanding of the nature of data models, particularly semantic data models. We seek to develop a greater ontological understanding of the data models themselves.

The semantics of mathematical structures excites very little controversy. In short, mathematical semantics usually arises in two forms which might be described as *absolute semantics* and *relative semantics*, both of which will be relevant for our treatment of data models below. In absolute semantic terms, the meaning of a mathematical object is fully determined by its interactions — by the operations it can perform and by the axioms which they satisfy. This was emphasized in the nineteenth century by the assertion that substituting any unused term consistently for any part of a mathematical structure (for example, beer jug for line in a geometry) changed the mathematical structure not at all, since the part (beer jug) was required to satisfy the same axioms (for example, any two points determine a unique beer jug and any two beer jugs intersect at a point). Relative semantics, on the other hand, permits a hierarchical structuring of semantics by allowing a new mathematical structure to be defined based on another, already assumed to be understood structure. Thus a topological group is defined exactly as a group is defined, but the definition is based on topological spaces (rather than sets) and correspondingly the operations must be continuous maps rather than mere functions.

There have been two principal approaches that mathematicians, if not necessarily philosophers, have taken to ontological issues in mathematics. Both are themselves branches of mathematics. The first is universal algebra and the second is category theory.

The objects of study in universal algebra are *signatures* of (specifications for) mathematical structures — the type theory, operations and axioms that will in the sense of absolute semantics fully determine the meaning of a mathematical structure — and the *varieties* which are the collection of all mathematical structures which match in a specified way the signature (the collection of all *models* of the signature). In absolute semantic terms a signature is an ontology for a variety, and universal algebra may be considered as a mathematical meta-ontology being the mathematics of the ontologies of varieties.

The methodology that we use in information systems consultancies is based on category theory and incorporates ideas from universal algebra. Category theory is a branch of mathematics renowned for its semantic power, its simple axiom set, and its use of graphical techniques. It has been widely used for specification in computer science in, for example, abstract data types [11], [12], semantics of programming languages [20], [21], and functional programming [1]. The basic notions of category theory are categories, functors, and natural transformations. In addition, universal constructions including limits and colimits and adjoints are fundamentally important. These notions will not be reviewed here, but can be found in any elementary category theory text including [2], [17], [23].

Categorical universal algebra is a category theoretic generalization of universal algebra, and is the principal inspiration for our information system research and development methodology. Categorical universal algebra replaces the notion of signature with a category usually called a *theory* or *classifying category*. Models of the classifying category $\mathcal{T}$ are certain functors into some category $\mathcal{C}$, and the corresponding full subcategory of the functor category $[\mathcal{T},\mathcal{C}]$ takes the place of a variety. Among other advantages this generalization yields the correct morphisms of the mathematical structures as the natural transformations between the models, and supports relative semantics by allowing the category $\mathcal{C}$ to be itself a category of models of some other classifying category. As before, this use of category theory can be viewed as meta-ontological.

How does all this relate to information systems research? Over a number of years we have been using the techniques of categorical universal algebra both for large scale industrial consultancies with for example a major oil company [6], a government department of health [5], and Australia's largest telecommunication company [4], and to obtain a clearer ontological understanding of information systems to allow us to develop new and mathematically precise treatments of difficult problems in the foundations of information systems.

The goal of this paper is to illustrate some of the practical advantages of an ontological understanding of information systems based on our category theoretic methodology. We choose a number of areas where we have obtained valuable new insights.

The plan of the paper is as follows. In Section 2 we begin with an example illustrating the category theoretic information system specification techniques that we use. In Section 3 we define views in our category theoretic framework. Section 4 explains how for these views the view update problem, a difficult and longstanding problem in database theory, can be analyzed. In Section 5 we show how the theory developed in the earlier sections can be used to discover and support synchronized data in extant information systems. Again this is an important problem. In Section 6 we point out that our category theoretic approach is an extension of common techniques for representing ontologies. It is useful to note this, as category theory can often seem daunting — in fact, most of the basic properties of categories are familiar to practitioners using common ontological tools. Finally, Section 7 distinguishes between prescriptive ontologies (standards) and small application based ontologies, and speculates that the category theoretic techniques may be particularly well-suited to the latter, where semantic mismatch needs to be managed locally and operations for combining ontologies (much as we combine interoperating systems) are important.

2. Category Theoretic Specification: An example

In this section we give an example to illustrate some of the category theoretic tools we use in our information systems consultancy work, and we introduce some basic terminology. The main problem that we will consider here is the *specification* of information systems, including particularly the specification of real-world constraints. The example that we have chosen is artificial, and it has been taken from the authors' [8]. It has been constructed so as to illustrate the main constraint specification tools with a minimum of extraneous details.

One of the important aspects of the example that we have chosen is that it illustrates the semantic modeling power of our approach by naturally incorporating constraints. If we are to accept the mathematicians' notion of absolute semantics, we still need to be

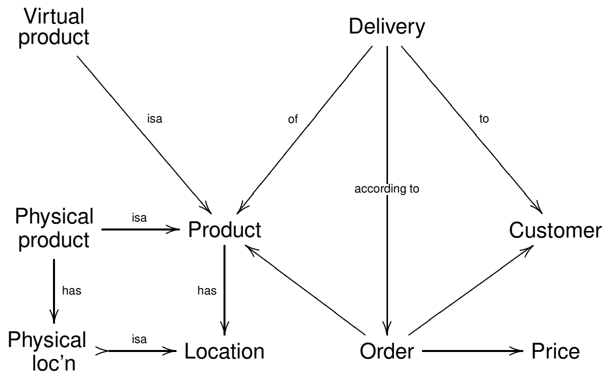

Fig. 1. An illustrative example: Part of an e-commerce schema

assured that the constraints that our methodology supports capture a large proportion of the constraints that are important in the domain we are modeling.

As noted above we will not review elementary category theoretic notions. Definitions and basic properties of commuting diagrams, limits, and coproducts, can be found in most introductions to category theory. We will also assume familiarity with ER terminology [22].

The graph shown in Figure 1 is a part of a specification for an e-commerce information system. The graph may be thought of as an ER model: The vertices of the graph represent entities (types), the arrows represent many-to-one mandatory relations (functions or methods), and the arrow with tail ($\rightarrowtail$) represents a sub-entity (subtype or injective function). There is a correspondence between standard ER models and such graphs given by tabulating many-to-many relations and using subentities to eliminate non-mandatory many-to-one relations. For simplicity attributes have not been shown in Figure 1, with the sole exception of Price, but if desired they can be included as arrows from the entity on which they are defined to a node representing their attribute value set, or they can be recorded in a separate data dictionary.

Of course, the specification technique we use also has extra-ER aspects. We specify three further sets of properties: Which diagrams commute, which nodes occur as limits of specified diagrams (via specified cones) defined in the graph, and which nodes occur as coproducts (via specified cocones) of specified nodes of the graph. In our example the extra-ER properties include the specification that the two triangles commute, that the square be a pullback (a kind of limit), and that the Product node be the coproduct of Physical product and Virtual product (electronically available software, music, etc). These constraints ensure that: Deliveries only take place to fill specific appropriate Orders (the commuting triangles); All Products, and only those Products, which have a Physical location are Physical products (the pullback); All Products are either Physical products or Virtual products (the coproduct). The Location entity stores both warehouse locations and URLs.

In our technique the quadruple consisting of the graph, sometimes called the *underlying graph*, and the extra-ER constraints (three sets), is called a schema. Formally

DEFINITION 1. A *schema* $\boldsymbol{E} = (G, \mathcal{D}, \mathcal{L}, \mathcal{C})$ is a directed graph G, a set $\mathcal{D}$ of pairs of paths in G with common source and target, a set $\mathcal{L}$ of cones in G, and a set $\mathcal{C}$ of cocones in G.

(It may be worth noting that the definition uses merely directed graphs — there is no need to decorate some arrows with tails to indicate subentities. This follows from the simple technical result that the property of being a subentity can also be specified as a limit. See for example [15] page 539.)

A schema can be thought of as the raw material, the base entities and relationships, for the *classifying category*. Roughly speaking it is the smallest category containing the schema, satisfying the constraints, and closed under finite limits and finite coproducts. The classifying category has objects corresponding to each of the structural queries that can be applied to the information system [7]. The classifying category for $\boldsymbol{E}$ is denoted by $Q(\boldsymbol{E})$ to remind us that it incorporates all the queries. The category $Q(\boldsymbol{E})$ is usually infinite, so it is not likely to be very useful in implementations, but because it embodies all the structural queries it has theoretical value as it represents all possible data that can be derived from an information system, and because it is finitely generated it remains tractable.

Schemata can be interrelated using schema maps. A *schema map* is a graph morphism between the corresponding graphs which maps each of the constraints on the first schema to a constraint (already) specified in the second schema. Formally

DEFINITION 2. Consider two schemata $\boldsymbol{E} = (G, \mathcal{D}, \mathcal{L}, \mathcal{C})$ and $\boldsymbol{E}' = (G', \mathcal{D}', \mathcal{L}', \mathcal{C}')$. A *schema map* $m : \boldsymbol{E} \longrightarrow \boldsymbol{E}'$ is a graph morphism, also denoted $m : G \longrightarrow G'$ which carries by composition diagrams in $\mathcal{D}$, cones in $\mathcal{L}$ and cocones in $\mathcal{C}$ to respectively diagrams in $\mathcal{D}'$, cones in $\mathcal{L}'$ and cocones in $\mathcal{C}'$.

Sometimes we will speak of a schema map from a schema into a category. In that case we mean a schema map from the schema into the underlying schema of the category: its graph is the underlying graph of the category, and its constraints are all of the constraints that happen to be true in the category — all of the commuting diagrams, all of the limits, and all of the coproducts.

An information system state, sometimes called a database state or instance, is an assignment of, for every node in the schema a finite set (the set of instances or values of that entity or attribute), and for every arrow in the schema a function between the corresponding sets (the relationships among the entity instances or the attribute values corresponding to the instances), such that the constraints are satisfied in the category of finite sets. Often we will elide the reference to database and simply refer to a database state as a state. Formally

DEFINITION 3. A *state* E for a schema $\boldsymbol{E}$ is a schema map $\boldsymbol{E} \longrightarrow \mathbf{Set}_0$ into the category of finite sets.

In other words, a state for a given schema is a diagram of (finite) sets and functions which is the same shape as the schema's underlying graph, and which satisfies the constraints.

This completes our small example, and our introduction of the basic terminology used in the category theoretic specification of information systems.

3. Relative semantics and views

Many information systems use *views* for a variety of purposes. A view mechanism allows data from a database to be selected and presented as if it were a (usually smaller) independent database. As a brief indication of the advantage of supporting views, we note that views can be used for security (by only providing access to data which need not be secure), privacy (by not providing all of the data, and a view doesn't even indicate that there is any "missing" data), maintenance (programs which access a database through a view will still work if the database is changed as long as the view mechanism is maintained), and relevance (a view can show what appears to be an entire database but is actually the result of a query which has selected records relevant to the expected user).

The question of which views are definable for a particular schema is long-standing, and has generated considerable controversy. The main problem has been that proposals for definable views have frequently been formulated to correspond to those view mechanisms which have been implemented in a particular system. Of course, such "definitions" are unlikely to be canonical. Instead we would want a mathematical definition of view which will be independent of particular system implementations.

A view might be thought of as an instance of relative semantics: The view as a database should be presented as a schema as usual but the meanings of objects in the view are determined by their relationship to the underlying database. This inspires the following definition.

DEFINITION 4. Let $\boldsymbol{E}$ be a schema. A *view* of $\boldsymbol{E}$ is a schema $\boldsymbol{K}$ and a schema morphism $k : \boldsymbol{K} \longrightarrow Q(\boldsymbol{E})$.

So a database state for a schema $\boldsymbol{K}$ is a functor from $\boldsymbol{K} \longrightarrow \mathbf{Set}_0$, while a view of $\boldsymbol{E}$ with schema $\boldsymbol{K}$ is a functor $\boldsymbol{K} \longrightarrow Q(\boldsymbol{E})$. Only the semantic category need change.

This definition captures arguably the broadest reasonable notion of views: Views can be constructed from any data which can be obtained from the database with schema $\boldsymbol{E}$ (since $Q(\boldsymbol{E})$ has objects corresponding to each of the structural queries definable over $\boldsymbol{E}$), and those data can be structured in any way ($\boldsymbol{K}$ is an arbitrary database specification schema), subject only to compatibility with the constraints of $\boldsymbol{E}$. Furthermore, basic properties of views, including view calculation and view composition, are well known category theoretic properties. In particular, views are the morphisms of the Kleisli category [2] of Q.

The phrase "compatible with the constraints of $\boldsymbol{E}$" is important: We have argued that those constraints help to capture the semantics associated with the entities of $\boldsymbol{E}$, and so it is of course very important that views have compatible, although not necessarily identical, semantics.

4. View updates and cartesian arrows

When a database state changes, the state of a view of the database changes automatically. In fact this can be seen precisely: A state is a certain functor or equivalently a schema morphism $Q(\boldsymbol{E}) \longrightarrow \mathbf{Set}_0$, and a view is a schema morphism $k :$ $\boldsymbol{K} \longrightarrow Q(\boldsymbol{E})$. Composing those morphisms gives a state of the view, and any changes in the first morphism result immediately in changes in the state of the view.

The operation *composing with k*, which takes states of the database to states of the view, is traditionally (in category theory) denoted k^*.

Since a view is intended to be treated as an apparently stand-alone database the converse problem of how a database state should be changed to remain synchronized with the state of a changing view is important. This is known as the view update problem.

There has also been considerable controversy over the view update problem, arguably for the same reason as the controversy over views: People have defined updatability based on what their systems have been implemented to do, rather than seeking a fundamental definition. In fact there is a real difficulty with obtaining a fundamental definition in the usual framework because traditionally people have sought to define updatability for schemata. Instead updatability may be treated as a property of the current *state* of the database, and then updatability can be seen to be a well-known category theoretic property (in the following definition we assume, for technical reasons, that the database is keyed, and we give the definition for insert updates — the delete update definition is dual).

DEFINITION 5. Suppose given a state D of a database with schema $\boldsymbol{E}$, and a view insert update $m : k^*D \rightarrowtail V'$, then this is an *updatable insert* if there is an opcartesian morphism at D over m. Furthermore, that opcartesian morphism is the required update of the database.

Once again this, in a sense which can be made mathematically precise, is the widest reasonable definition of view update. Furthermore, the definition is relatively easy to apply since elementary category theoretic properties allow us to prove that all states of various classes of views are updatable (more details can be found in [15]).

5. Interoperation

Now that we have a mathematical understanding of views and view updates, we are in a position to develop applications of those ideas taking advantage of mathematical results about view updates. One of the most important applications is to the development of interoperations between extant information systems.

In order to design system interoperations we develop a single schema $\boldsymbol{K}$ together with two schema morphisms which exhibit $\boldsymbol{K}$ as a view of each of the information systems. The schema $\boldsymbol{K}$ is a single formal representation of those parts of the systems that share common data in whatever form. As noted above, operations of the form k^* ensure that changes in either system result in changes of the view. Provided that we can solve the view update problem for those changes with respect to the other system, then we have achieved interoperation. Indeed, the proof that the view update problem is solved for the circumstances in question is a formal statement of the interoperations code [14].

This application depends vitally upon the generality of the definition of view (so that any data in any form can occur in a view), and on the definition of updatability which is designed to ensure that views which are not updatable are not updatable in principle (rather than just because such updates have not been implemented).

6. Ontologies: Trees, graphs, and categories

Having reviewed some of the useful research results obtained by using category theory as a meta-ontology, we speculate briefly about where category theory might stand among ontological representation techniques.

Many information system ontologies are trees. The ontologies provide a language by defining terms, the nodes of the trees, and their inter-relationships. Indeed, since many grammars can be specified using trees or TAGs, there is a very close relationship between many formal linguistic representations of a domain, and tree based representations. The work of Dejima Corporation [10] provides just one novel example: They develop ontological representations of application domains as trees, and then, populating the nodes with recognizing agents, analyze domain text by allowing agents to compete for text that the agents seem to recognize. Each apparent recognition is passed up the tree to further agents, and winning recognitions are those that are consistent across a significant area of the tree.

Of course when the number of relationships among terms grows large, it is unlikely that a tree structure can easily be maintained. In this case graphs (usually directed) play a similar but more flexible role. For a relevant example see the work of Wiederhold and the SKC group, for example [18]. Much of their work is directed towards obtaining an algebra of ontologies, a point that we will return to below.

We take the point of view that the edges in ontological structures like trees and graphs represent relations which can be composed. In many examples this is so obvious as to appear to appear vacuous: Of course if X is a part of Y which is a part of Z, then X is a part of Z. But if the structure is a graph there may be multiple paths between two nodes, and their equality or otherwise once composed is an important question and can be particularly revealing in studying the semantics of the entities represented. For example, very many business rules are represented by commuting diagrams [13].

Thus, a category, which can be presented as a graph together with a specification of commutative diagrams, is a minor, albeit powerful, extension of common ontological representations.

7. Ontologies: Prescriptions versus semantic representations

There are at least two broad approaches taken to the ongoing development of domain ontologies. In the first, which we describe as prescriptive, large organisations develop standards that allow participants in their industries to work within an agreed ontology. This supports system interoperation very well, but limits flexibility. In the second, which we describe as semantic, application ontologies are developed relatively freely, but with detailed semantic information, so that organisations which wish to interoperate can consult the detailed semantic information to determine whether apparent matches in two different ontologies are semantically justified (the semantically inconsistent uses of even common terms such as "employee" or "product" are notorious, and rife).

In both cases there is considerable value in having an *algebra of ontologies*. Such an algebra provides operations that, analogously with union and intersection, combine two already extant ontologies to give a new ontology including each of them, or to give a new ontology of the terms common to each of them. Some groups are actively pursuing the development of such an algebra, while others are arguing that it is an unattainable ideal that will founder on semantic mismatch.

Where does the category theoretic approach lie in all of this? While not incompatible with the development of standards, it seems best suited to the semantic approach to ontology development. We would argue that, as in our information systems experience, the detailed semantic information required is built into the category theoretic representation using constraint specifications including limits and colimits. Our experience in developing systems interoperations suggests that the tools for resolving semantic mismatch already reside in the classifying categories, and the extra semantic information plays an important role. Furthermore, category theory itself provides an algebra for categorically specified ontologies, including important operations like pushout and equaliser as well as coproduct and product.

It is too early to tell how useful category theory might be for the specification and maintenance of ontologies in general, but our experience of category theory as a (meta) ontology for information systems research indicates that it warrants detailed exploration.

8. Conclusion

This paper has reviewed a number of examples where the deeper understanding of the ontological issues surrounding data modeling, viewed as a mathematical specification problem, has led to insights which have been commercially valuable, and it has in the last two sections speculated on why this approach to ontology has been useful.

We repeat that there are important ontological issues in the application domain, and that we have not addressed them at all. Rather we have concentrated on bringing insights from mathematical ontology to bear on practical problems in information systems research and development. All of the results reported above have depended fundamentally upon the use of category theory as a meta-ontology for information systems research, on classifying categories as an ontology for particular information systems, and on the strength of the category theoretic methodology as a semantic data model.

A question which naturally arises is: How wide is the range of constraints that can be represented using category theoretic schema? The answer is very wide indeed, and just how wide can be made mathematically precise. Techniques from categorical logic can be used to show that a very large subset of all first order logical constraints can be modeled by category theoretic schemata. That subset includes all of the constraints that the authors have ever needed in practical applications.

One final observation: While the notion of mathematical ontology that we have employed has been fundamentally dependent on the absolute semantics common in mathematics, in the body of the paper we have made little explicit mention of relative semantics. So it is worth repeating that views are an important example of relative semantics. Further examples related to the foundations of information systems are presented in [16] where for any given information system the category of structural queries, the category of states, and the insert/delete dynamics are each shown to be obtained by different relative semantics for the one ontology (classifying category).

ACKNOWLEDGMENTS

The research reported here has been supported in part by the Australian Research Council and by the NSW Department of Health. The authors are grateful for the assistance of their colleagues Robert Colomb, Jeannine Hammersley, and Robert Rosebrugh.

References

[1] A. Asperti and G. Longo. *Categories, Types and Structures: An introduction to category theory for the working computer scientist*. MIT Press, 1991.

[2] M. Barr and C. Wells. *Category theory for computing science*. Prentice-Hall, second edition, 1995.

[3] Robert Colomb, C. N. G. Dampney and Michael Johnson. The use of category-theoretic fibration as an abstraction mechanism in information systems. Accepted for *Acta Informatica*, 2001.

[4] C. N. G. Dampney and Michael Johnson. TIME Compliant Corporate Data Model Validation. Consultants' report to Telecom Australia, 1991.

[5] C. N. G. Dampney and Michael Johnson. Fibrations and the DoH Data Model. Consultants' report to NSW Department of Health, 1999.

[6] C. N. G. Dampney, Michael Johnson and G. M. McGrath. Audit and Enhancement of the Caltex Information Strategy Planning (CISP) Project. Consultants' report to Caltex Oil Australia, 1993.

[7] C. N. G. Dampney, Michael Johnson, and G. P. Monro. An illustrated mathematical foundation for ERA. In *The unified computation laboratory*, Eds C. Rattray and R.G. Clarke, Oxford University Press, 77–84, 1992.

[8] C. N. G. Dampney and Michael Johnson. Enterprise Information Systems: Specifying the links among project data models using category theory. Proceedings of the International Conference on Enterprise Information Systems, 619–626, 2001.

[9] C. N. G. Dampney, Michael Johnson, and Robert Rosebrugh. View Updates in a Semantic Data Model Paradigm. Proceedings of ADC2001, IEEE Press, 29–36, 2001.

[10] Dejima Corporation. `www.dejima.com`

[11] J. Goguen, J. W. Thatcher and E. G. Wagner. An initial algebra approach to the specification, correctness and implementation of abstract data types. In R. Yeh, *Current Trends in Programming Methodology IV*, 80–149, Prentice-Hall, 1978.

[12] J. Goguen, J. W. Thatcher, E. G. Wagner, and J. B. Wright. Initial algebra semantics and continuous algebras. *JACM*, 24:68–95, 1977.

[13] Michael Johnson and C. N. G. Dampney. On the value of commutative diagrams in information modelling. In the Proceedings of AMAST'93, *Springer Workshops in Computing*, 1994.

[14] Michael Johnson and Robert Rosebrugh. Database interoperability through state based logical data independence. *International Journal of Computer Applications in Technology*, in press, 2001.

[15] Michael Johnson and Robert Rosebrugh. View updatability based on the models of a formal specification. Proceedings of Formal Methods Europe, LNCS 2021, 534–549, 2001.

[16] Michael Johnson, Robert Rosebrugh, and R. J. Wood. Entity-relationship models and sketches. Submitted, 2001.

[17] Saunders Mac Lane. *Categories for the Working Mathematician*. Graduate Texts in Mathematics 5, Springer Verlag, 1971.

[18] Prasenjit Mitra, Gio Wiederhold and Martin Kersten. A Graph-Oriented Model for Articulation of Ontology Interdependencies. In the Proceedings of the Conference on Extending Database Technology, (EDBT 2000), Konstanz, Germany, March, 2000. Also, Stanford University Technical Note, CSL-TN-99-411, August, 1999.

[19] John Myopoulos. Next generation database systems won't work without semantics! Panel session, *SIGMOD Record*, 27:497, 1998.

[20] D. Scott. Continuous lattices. *Lecture notes in Mathematics*, 274:97–136, 1972.

[21] D. Scott. Domains for denotational semantics. *Lecture notes in Computer Science*, 140:577–613, 1982.

[22] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*. Volume 1, Computer Science Press, 1988.

[23] R. F. C. Walters. *Categories and Computer Science*. Cambridge University Press, 1991.