The Unified Computation Laboratory. C M I Rattray & R G Clark (eds)

An Illustrated Mathematical Foundation for ERA

C.N.G Dampney, Michael Johnson and G.P. Monro

School of Mathematics and Computing, Macquarie University
and School of Mathematics and Statistics, Sydney University
AUSTRALIA

Abstract

This paper outlines an application of category theory to provide mathematical foundations for the ERA (Entity, Relationship, Attribute) approach to information systems. An ERA specification yields the *dynamic category* **DC** of the information system, which is a syntactic object describing all allowable operations. The dynamic category can be "collapsed" to a *static category* **SC**, which has a canonical language associated with it. This canonical language is the *query language* of the information system. A *model* of the information system is defined to be a functor from **DC** to the category **Set** and "snapshots" of a model are described in terms of functors from **SC** to **Set**.

1 Introduction

In the last three decades the study of syntax and semantics has benefited greatly from category theory [7], [5]. A formal theory has associated with it a category (generally a "free" category in an appropriate sense). We call this category the *syntactic category* of the formal theory. A *model* of the theory (i.e. an "algebra") is a functor from the syntactic category to the category **Set** of sets and functions. An implementation (or interpretation) of one theory in another becomes a functor between the appropriate syntactic categories. The concept of functor thus unifies several different notions. As this example shows, category theory is able to unify previously disparate notions and provide a solid theoretical foundation for them.

In this paper we outline a category-theoretic description of the syntax and semantics of the ERA (Entity, Relationship, Attribute) approach to information system specification [2] (including database specification). An ERA specification yields two syntactic categories, a "dynamic category", and, derived from the dynamic category, a "static category". The static category in turn yields the *query language* for the information system. Models of the information system are described in terms of functors from the two syntactic categories into **Set**.

The work described here is still at a preliminary stage. However we believe that the framework that we are developing will provide an appropriate theoretical foundation for the ERA approach. We also believe that it will provide a foundation that can be extended to cover other approaches to information system design, such as Jackson's Communicating Sequential Processes (CSP) method [4]. (Dampney [3] has explored some possible transformations between ERA and CSP but a full treatment of their interrelationship requires a firm mathematical foundation.) The overall aim is to bring as many approaches as possible together into a coherent whole which can guide the development of database specification systems.

2 A little category theory

The standard textbook on category theory is [6], which could be consulted for more details. There are several newer books including Barr and Wells [1] and Walters [8] which introduce category theory via its applications to computer science. These latter may be more appropriate for readers with a background in computing.

A *category* consists of a set of *objects* and a set of *morphisms* (also called *arrows*). A morphism f from the object A to the object B is written $f : A \to B$. Morphisms compose associatively and there are identity morphisms for the composition.

Two basic constructions in categories are those of *product* and *coproduct*. The product of two objects A and B is written $A \times B$ and generalizes the notion of *cartesian product* of sets. The coproduct of A and B is written $A + B$ and generalizes the notion of *disjoint union* (discriminated union) of sets. The categories we deal with also have an initial object $\mathbf{0}$ and a terminal object $\mathbf{1}$.

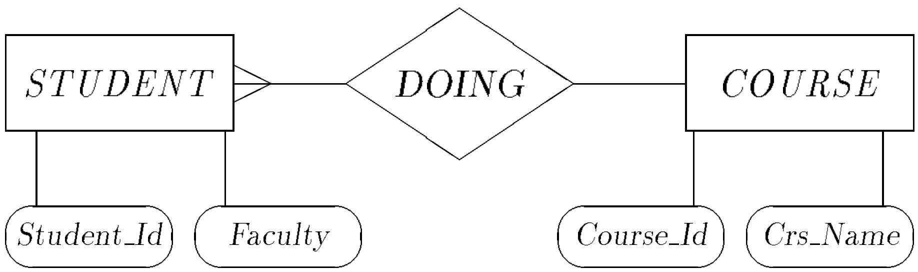
In syntactic categories we can think of the objects as types; $\mathbf{0}$ corresponds to an empty type and $\mathbf{1}$ a type containing only one thing. The object $\mathbf{1} + \mathbf{1}$ (the coproduct of $\mathbf{1}$ and $\mathbf{1}$) will be called **Bool**, and corresponds to the usual Boolean type. Our categories allow the construction of subtypes; if $F : A \to \mathbf{Bool}$ is a morphism, there is a corresponding subtype of A which is written as $\lceil F \rceil \to A$. Intuitively this is the subtype of A containing those elements a of A such that $F(a)$ is true. There is also the subtype $\lceil \neg F \rceil$ containing those elements a of A such that $F(a)$ is false. We will frequently require subtypes of products: if $G : X \times Y \to \mathbf{Bool}$, there are corresponding subtypes $\lceil G \rceil$ and $\lceil \neg G \rceil$ of $X \times Y$.

The categories we use have sufficient structure so that they have canonical languages associated with them. That is to say, the categories are *coherent categories*, which are the same as the *logical categories* of [7]. We also require our categories to have further structure: notably every object is to have a decidable equality relation and decidable equivalence relations are to have coequalizers.

The logic associated to such a category is what is called *coherent logic:* this is not quite full first-order logic, but is powerful enough to express database operations and queries in the categories we consider.

3 ERA specifications

We give here a very simple example of an ERA specification. There are two entities, *COURSE* and *STUDENT*. *COURSE* has the attributes *Course_Id* and *Course_Name*. *STUDENT* has the attributes *Student_Id* and *Faculty*. There is one *relation DOING*, which indicates which course (if any) a student is currently taking. The resulting ERA diagram is as follows.



This database is intended to reflect current information only, so that (for example), a student who leaves the institution will be deleted from the database. A course may be taken by many students, or by none. We will assume (as usual) that one of the attributes of an entity serves as a unique identifier for instances of that entity. For *STUDENT* this unique identifier is *Student_Id*.

Each *attribute* is considered to range over a fixed set of values (so for example *Student_Id* might range over the set of all eight-digit numbers). Of course at any one time most of these numbers do not have students assigned to them, but we need to consider the fixed range for these numbers in our theory. However instances of *entities* can be created and destroyed; for example students can enter or leave the database, so the number of students will fluctuate with time. Courses can also be created or destroyed.

An ERA specification determines a "dynamic category", which describes the total dynamics of the database. The dynamic category in turn gives rise to a static category, which in its turn gives the *query language* for the database. In the following sections we describe the dynamic and static categories, starting with the static category (since it is simpler).

4 The static category

Like all categories, the static category consists of objects and morphisms. We describe some of the objects and morphisms in the category; the rest of the category can be thought of as the closure of the given objects and morphisms under the constructions of product, coproduct, and so on. In particular the static category will always contain the objects $\mathbf{1}$ and $\mathbf{1} + \mathbf{1} = \mathbf{Bool}$.

For each entity, we introduce an object (which we call the *entity object* for that entity). For each attribute of an entity we have a corresponding *attribute object*, and a morphism from the entity object to the attribute object (ERA practitioners usually think of this morphism as the attribute). For each relation between two entities we have a morphism from the product of the two entity objects to **Bool**. The example in Section 3 gives rise to the following:

$$Student\_Id \longleftarrow Student \longrightarrow Faculty$$

$$Course\_Id \longleftarrow Course \longrightarrow Course\_Name$$

$$Doing : Student \times Course \to \mathbf{Bool}.$$

Since an attribute like *Faculty* maps to a fixed value set, we build this into the theory. If (for example) there are five faculties, the attribute object for *Faculty* is the coproduct $\mathbf{1}+\mathbf{1}+\mathbf{1}+\mathbf{1}+\mathbf{1}$. (If the number of faculties is variable then *Faculty* should be an entity, not an attribute.) This coproduct construction means that each individual faculty has a morphism $\mathbf{1} \to Faculty$ representing it in the category.

The static category has a canonical language which serves as the query language for the database. This query language admits all the usual queries for relational databases. For example, since we included a morphism which is effectively the name of each faculty, we can ask for all the students in a named faculty. Constructions like the join of two relations are represented in the static category, and can be expressed in the canonical language. This query language is not an *ad hoc* construction, but arises naturally from the static category. Indeed the close connection between languages and categories is characteristic of the sort of category theory used in this subject.

5 The dynamic category

The static category described in the previous section is derived from a considerably more complicated dynamic category. This dynamic category has to allow for all possible update operations on the database.

Again the dynamic category is constructed by specifying certain objects and morphisms. The actual dynamic category is then obtained as the closure of the given objects and morphisms under the various operations appropriate to these categories.

For each entity we assign an object, and for each attribute we assign an object. For *STUDENT* we have the three objects *SSTUDENT*, *Student_Id* and *Faculty*. The unique identifier attribute *Student_Id* is given a special role in the dynamic category, through the following morphisms. (The effect is that students are normally identified by their student numbers.)

$New_S : \mathbf{1} \to SSTUDENT$
$In_S : SSTUDENT \times Student\_Id \to \mathbf{Bool}$
$Add : \lceil \neg In_S \rceil \times Faculty \to SSTUDENT$
$Delete : SSTUDENT \times Student\_Id \to SSTUDENT$
$Fac\_Read : \lceil In_S \rceil \to Faculty$

Intuitively (that is, in the intended interpretation) *SSTUDENT* is the set of sets of students which might be in existence at any one time, where each student is "tagged" with the student's faculty (and any other attribute of students). The extra S in the name of the object *SSTUDENT* is to remind us that it is to be thought of as a *set of sets* of students. The various morphisms just given have the following interpretations.

New_S creates a new database (with no students in it as yet).

In_S tests whether or not a possible student identity number has a student assigned to it. Note that the morphism In_S determines subtypes $\lceil In_S \rceil$ and $\lceil \neg In_S \rceil$ of $SSTUDENT \times Student\_Id$ (as explained in Section 2).

Add adds a new student to the database. The user supplies the student number and faculty, but we have built in to the operation the restriction that a currently unassigned student number is to be used. (If there were attributes additional to *Faculty* these would also have to be specified when the student is added.)

Delete deletes a student entirely from the database.

Fac_Read gives the faculty of existing students (intuitively, given a pair consisting of a set of students and an identity number of a student in the set, *Fac_Read* gives the faculty of the student with that identity number).

We add appropriate axioms to the dynamic category to ensure that the various operations just described function as expected.

Note that once a student has been added to the database, its attributes cannot be changed. However a student can be deleted entirely from the database, and then a new student can be added with the same student number but a different faculty (for example).

As with *STUDENT*, the entity *COURSE* will have associated with it in the dynamic category objects *SCOURSE*, *Course_Id*, etc, and morphisms

$New_C : \mathbf{1} \to SCOURSE$

$In_C : SCOURSE \times Course\_Id \to \mathbf{Bool}$, etc.

For *DOING* we have an object *SDOING* and two morphisms

$$base_S : SDOING \to SSTUDENT \quad \text{and} \quad base_C : SDOING \to SCOURSE.$$

Intuitively *SDOING* is a set of possible relationships where each relation describes a snapshot at a time instant in which a particular set of students is doing a particular set of courses. Thus each relation has its own set of current students and current courses. The morphisms $base_S$ and $base_C$ serve to identify the set of current students and the set of current courses for a particular relation in *SDOING*. That is, if $R \in SDOING$, $base_S(R)$ is the set of current students for R and $base_C(R)$ is the set of current courses for R. We stress that this is the intuitive meaning of *SDOING*: formally *SDOING* is merely an object in a category, not a set of relations.

We now form the following diagram, where we use the abbreviations D for *SDOING*, S for *SSTUDENT*, C for *SCOURSE*, *Sid* for *Student_Id*, *Cid* for *Course_Id*, and *Fac* for *Faculty*. The arrows at the base of the diagram are the products of

$base_S$ and $base_C$ with identity arrows.

$$
\begin{tikzcd}
 & & XD \arrow[dl] \arrow[dr] \arrow[d, phantom, "\mathrm{pb}"] & & \\
\lceil In_S \rceil \times Cid \arrow[d] & P_1 \arrow[l] \arrow[dr] & {} & P_2 \arrow[r] \arrow[dl] & \lceil In_C \rceil \times Sid \arrow[d] \\
S \times Sid \times Cid & & D \times Sid \times Cid \arrow[ll] \arrow[rr] & & C \times Cid \times Sid
\end{tikzcd}
$$

Here the left and right squares are pullbacks, so the objects P_1 and P_2 arise by pullback; the middle diamond then defines the object XD by pullback. XD is a sort of unpacked version of *SDOING*. Intuitively it is the set of triples (R, s, c), where R is a relation between students and courses, s is a student in $base_S(R)$ and c is a course in $base_C(R)$.

We now add a morphism

$$Does\_course : XD \to \mathbf{Bool}.$$

This morphism tells us whether or not a given (current) student is doing a given (current) course. The complexity of things like XD is needed to take account of the fact that the set of current courses and the set of current students are variable.

We finally add morphisms $D \times Sid \times Cid \to D$ allowing us to assign a student to a given course and to unassign a student, and appropriate axioms controlling them. We also need axioms to say (for instance) that a student who has been newly added to the database is not enrolled in any course.

We end up with a collection of objects and morphisms, and axioms on these; the axioms can be expressed as equations between morphisms. The dynamic category is then constructed as the closure of these data under the constructions of coherent categories.

The static category is obtained from the dynamic category by a process of collapsing: we replace the objects *SSTUDENT*, *SCOURSE* and *SDOING* by the object $\mathbf{1}$. This causes consequential changes throughout the category. (In particular the object $\lceil In_S \rceil$ in $\mathbf{DC}$ becomes the object *Student* in $\mathbf{SC}$.) We will call the dynamic category $\mathbf{DC}$ and the static category $\mathbf{SC}$. The collapsing process defines both the category $\mathbf{SC}$ and a functor $\mathbf{DC} \to \mathbf{SC}$.

Just as the static category had a language associated with it (the query language), so also does the dynamic category have an associated language. This language is richer than the query language, as it can describe changes to the database.

6 Semantics

The category-theoretic way of approaching semantics is as follows. If we have a syntactic category C, a *model* or *interpretation* of C is a functor $M : C \to \mathbf{Set}$. In our situation C is a coherent category, and we require M to be a so-called coherent functor. This means that M takes products, coproducts, and similar constructions in C to the corresponding constructions in $\mathbf{Set}$.

In the situation of this paper we have two syntactic categories, the dynamic category $\mathbf{DC}$ and the static category $\mathbf{SC}$. A *model* of the ERA specification is a coherent functor $M : \mathbf{DC} \to \mathbf{Set}$. So (in our example with five faculties) $M(\mathit{Faculty})$ is a set containing five elements, and $M(\mathit{Student\_Id})$ is a set with 10^8 elements. $M(\mathit{SSTUDENT})$ is also a set, but it is a set each of whose elements can be regarded as a set of students. That is, $M(\mathit{SSTUDENT})$ is the set of *all possible sets of students* in the model. Similarly $M(\mathit{SCOURSE})$ is the set of all possible sets of courses, and $M(\mathit{SDOING})$ is the set of all possible relations in the model between a possible set of students and a possible set of courses. Furthermore, $M(\mathit{Add})$ is the function which implements the process of adding a student to the database and so on.

A *snapshot* (or instantaneous picture) of the model M is a functor $S : \mathbf{SC} \to \mathbf{Set}$ together with a natural transformation α as shown in the following diagram.

$$\begin{tikzcd}
\mathbf{DC} \arrow[dr, "M"] \arrow[d] & \\
\mathbf{SC} \arrow[r, "S"'] \arrow[u, phantom, ""] & \mathbf{Set} \arrow[l, phantom, shift left=4, "\alpha \Uparrow"]
\end{tikzcd}$$

S is a model of the static category $\mathbf{SC}$ and so is an interpretation of the query language. The natural transformation α ensures that S is a snapshot *of the model* M (that is, the model S fits properly into the bigger model M).

7 Acknowledgement

The authors gratefully acknowledge the support of the Australian Research Council (ARC).

References

[1] M. Barr and C. Wells, *Category theory and computer science*, Prentice Hall, 1990.

[2] P.P.S. Chen, The entity relationship model—towards a unified view of data, *ACM Transactions on Database Systems*, **1**, 9–36, 1976.

[3] C.N.G. Dampney, From ERA to CSP denotations: a synthesis of information system static and dynamic models, *Macquarie University Computing Report*, 1989.

[4] M. Jackson, *Jackson System Development*, Addison Wesley, 1982.

[5] F.W. Lawvere, Functional semantics of algebraic theories, *Proceedings of the National Acadademy of Sciences U.S.A.*, **50**, 869–871, 1963.

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

[7] Michael Makkai and Gonzalo E. Reyes, First Order Categorical Logic, *Lecture Notes in Mathematics*, **611**, Springer-Verlag, 1977.

[8] R.F.C. Walters, *Categories and computer science*, Carslaw Publications, 1991 (to appear).

Computing Discipline
Macquarie University
N.S.W. 2109
AUSTRALIA

Department of Pure Maths
University of Sydney
N.S.W. 2006
AUSTRALIA