INFORMATIQUE THÉORIQUE ET APPLICATIONS

E. LIPPE
A. H. M. TER HOFSTEDE

A category theory approach to conceptual data modeling

Informatique théorique et applications, tome 30, n° 1 (1996), p. 31-79
<http://www.numdam.org/item?id=ITA_1996__30_1_31_0>

Informatique théorique et Applications/Theoretical Informatics and Applications
(vol. 30, n° 1, 1996, pp. 31 à 79)

A CATEGORY THEORY APPROACH TO CONCEPTUAL DATA MODELING (*)

by E. LIPPE ([1]) and A. H. M. TER HOFSTEDE ([1])

Communicated by G. LONGO

Abstract. – *This paper describes a category theory semantics for conceptual data modeling. The conceptual data modeling technique used can be seen as a generalization of most existing conceptual data modeling techniques. It contains features such as specialization, generalization, and power types. The semantics uses only simple category theory constructs such as (co)limits and epi- and monomorphisms. Therefore, the semantics can be applied to a wide range of instance categories, it is not restricted to topoi or cartesian closed categories. By choosing appropriate instance categories, features such as missing values, multi-valued relations, and uncertainty can be added to conceptual data models.*

Résumé. – *Cette contribution décrit une sémantique fondée sur la théorie des catégories et développée en vue d'une modélisation conceptuelle des données. On peut concevoir la technique ici utilisée de modélisation conceptuelle des données comme une généralisation de la plupart des techniques existantes de cette modélisation. Elle comprend des caractéristiques comme la spécialisation, la généralisation et des types des ensembles. La sémantique n'utilise que des constructions simples provenant de la théorie des catégories, comme les (co-)limites ainsi que les épi- et les monomorphismes. Par conséquent, elle peut s'appliquer à un large éventail de catégories exemplaires; elle n'est pas limitée aux topoi ou aux catégories cartésiennes fermées. En choisissant des catégories exemplaires appropriées, aux modèles conceptuels des données peuvent être ajoutées certaines caractéristiques comme les valeurs absentes, les relations à plusieurs valeurs et même l'incertitude.*

1. INTRODUCTION

When developing information systems the first crucial step is to develop a model that describes the problem domain. This, however, is easier said than done, as communication with problem owners can be notoriously difficult and data can be of a complex nature. To improve this modeling process, data modeling techniques have been introduced.

(*) Received January 13, 1995.

([1]) Computing Science Institute, Faculty of Mathematics and Informatics, University of Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, e-mail: {ernstl,arthur}@cs.kun.nl
Correspondence to A. H. M. ter Hofstede.

Informatique théorique et Applications/Theoretical Informatics and Applications
0988-3754/96/01/$ 4.00/

In the past decades, data modeling techniques have shifted from implementation oriented (such as the *network model* [5] and the *relational model* [12]) to problem oriented. Some examples of problem oriented data modeling techniques are the Entity-Relationship Model (ER) and its many variants (*see* e.g. [11]), functional modeling techniques, such as FDM [27], and object-role modeling techniques, such as NIAM [24].

Complex application domains, such as hypermedia and CAD/CAM, have led to the introduction of advanced modeling concepts, such as those found in the various forms of Extended ER (*see* e.g. [33], [13]), IFO [2], and object-role modeling extensions such as FORM [17] and PSM [19].

Problem oriented data modeling techniques are usually referred to as *conceptual* data modeling techniques, a name which reflects an intention to abide by the Conceptualization Principle [14]. This principle states that data models should deal only and exclusively, with aspects of the problem domain. Any aspects irrelevant to that meaning should be avoided. Examples of conceptually irrelevant aspects include aspects of (external or internal) data representation, physical data organization and access, as well as all aspects of particular external user representations such as message formats and data structures.

One of the most difficult aspects of modeling is to capture the precise intentions of the problem owners. Communication with the problem owners is therefore essential and must be supported by the data modeling technique used. This is the reason why conceptual data modeling techniques offer graphical representations of their underlying concepts: as this facilitates understanding of the models. A lack of graphic representation, and the lack of certain specialized concepts, explains why formal specification languages, such as Z [30] and VDM [20], never gained much popularity in this particular field of study. This emphasis on graphic representations has, however, a side-effect, building sound, formal foundations has been neglected.

This lack of formal foundations in the field of information systems has led to a situation referred to as the "Methodology Jungle" [1]. In [9] it is estimated that during the past years, hundreds, if not thousands, of information system development methods have been introduced. Most organizations and research groups have defined their own methods. Hardly any of these has a formal syntax, let alone a formal semantics. Discussion of numerous examples, many using pictures, is a popular style for the "definition" of new concepts and their behavior. This has led to *fuzzy* and *artificial* concepts in information systems

development methods and, to some extent, in conceptual data modeling techniques.

The goal of this paper is to define a general and formal framework for conceptual data modeling techniques. This framework should clarify the precise meaning of fundamental data modeling concepts and offer a sufficient level of abstraction to be able to concentrate on this meaning and avoid the distractions of particular mathematical representations (in a sense, the Conceptualization Principle can also be applied to mathematical formalizations). These requirements suggest category theory as an excellent candidate. Category theory provides a sound formal basis and abstracts from all representational aspects. Therefore, the framework will be embedded in category theory.

The framework described may also be of use for conceptual data modeling techniques that do have a formal foundation, as it may suggest natural generalizations and expose similarities between seemingly different concepts. The category theory foundation may also reduce proof obligations, due to the principle of duality.

Another interesting application of the use of category theory can be found in the opportunity to consider several different categories as semantic target domains. The categories that act as semantic target domains are called *instance categories*. For example if one wants to study uncertainty in a particular data modeling technique, it is natural to consider **FuzzySet**, the category of fuzzy sets, as an instance category. In this sense, the approach outlined is more general than approaches described in [32], [8] where only topoi are possible instance categories. Interesting categories, such as **Rel** and **FuzzySet** [25], [6] are not topoi.

Not every category can be used as instance category for data models. Appropriate categories should allow for certain constructions (e.g. coproducts should always be defined) and they should have certain special properties (e.g. coproducts have to be disjoint). Further, all populations representable in the category **FinSet**, consisting of finite sets and total functions, should be representable in an instance category. The category **FinSet** provides a kind of minimal semantics for data models. This category represents the intuitive (and most standard) semantics of data models. Therefore, definitions are mostly illustrated in terms of this category. Other categories will, of course, also be considered (e.g. **Rel**, **PartSet**, **FuzzySet**).

The paper is organized as follows. Section 2 contains an informal introduction to the important concepts of conceptual data modeling and

a general definition of the syntax. Section 3 presents mathematical results and notations needed for section 4, which focuses on the category theory definition of the type constructs discussed in section 2. A category theory definition of two important and frequently occurring constraint types in data models is the topic of section 5. Section 6 addresses the requirements that must be imposed on categories to be valid instance categories.

2. CONCEPTUAL DATA MODELING

In information systems applications capturing the data perspective turns out to be the main challenge for successful implementation. Problem domains in this field of study generally can be characterized as *data intensive*, computationally complex operations hardly occur. As remarked before, conceptual data modeling techniques aim at a precise, implementation independent, description of the data perspective of an application, which can be easily used in the communication with the problem owners. Formal transformations exist for translating conceptual data models to implementation oriented data models, underlying various relational or object-oriented database management systems (*see* e.g. [15]).

This section provides those readers not familiar with conceptual data modeling with an overview of the main type construction mechanisms and constraint types. The graphical representations of one technique, PSM [19], [18], will be employed, to avoid as much confusion as possible. PSM is sufficiently general for this purpose as it seems to contain all the essential concepts needed for conceptual data modeling. Actually, our current approach is even more general than PSM, since several restrictions in PSM have been removed. This section concludes with the definition of a type graph, which captures the formal syntax of conceptual data models. The semantics of a conceptual data model is the set of its possible instantiations, referred to as populations. Populations are only addressed informally in this section, a formal definition is given in section 4.

2.1. Basic object types

In conceptual data modeling techniques, a distinction is made between objects that have a structure and objects that do not have a structure. The former are discussed in sections 2.2 and 2.3. The latter are the so-called *basic objects*. Examples of basic objects are *persons*, *projects*, *names*, *project-codes*, etc. Usually a further distinction is made between objects that can

be represented directly on a communication medium (e.g. *names* or *project-codes*) and objects that cannot be represented directly on such a medium (e.g. *persons* and *projects*). The latter depend for their representation on the former. This issue is referred to as *identification*. In a conceptual data model each object should be identifiable, *i.e.* be denotable in terms of objects that can be represented directly. Persons for example could be identified by their names, while projects could be identified by a project-code. Of course, identification could be more complex. For certain applications it is conceivable that a more sophisticated identification scheme is needed for persons, as two different persons could have identical names. Identifiation plays a crucial role in conceptual data modeling as ultimately all relevant information has to be stored on a communication medium.

Basic objects can be instances of basic object *types*. A basic object type *Person* can for example have instances P_1 and P_2 at a certain point of time. An assignment of instances to an object type is referred to as a *population* of that object type. An assignment of instances to each of the object types in a conceptual data model is referred to as a population of that conceptual data model. Populations change in time as new objects may become relevant or existing ones may become irrelevant.

2.2. Fact types

One of the key concepts in data modeling is the concept of relationship type, sometimes also referred to as *fact type* (the preferred term in this paper). Generally, a fact type is considered to represent an association between object types. A graphic representation of a binary fact type R between basic object types A and B is shown in the PSM style in figure 1. In general relationships may be n-ary, where $n \geq 1$.

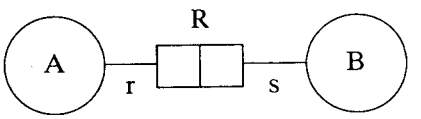


Figure 1. – Representation of a binary fact type

A concrete example of a binary fact type is given in figure 2. This schema captures the fact that *Persons* may be enrolled in *Courses*. A sample population of fact type *Enrollment* is shown in figure 3. In this sample population person P_1 is enrolled in the courses *CS114* and *CS115*, while person P_2 is enrolled in course *CS114* only.

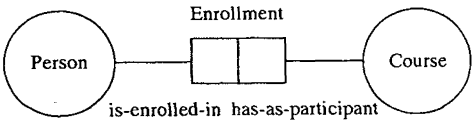


Figure 2. – A concrete example of a binary fact type

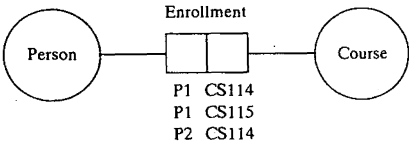


Figure 3. – A sample population of a fact type

A fact type consists of a number of *roles* (r and s in figure 1), denoting the way object types participate in that fact type. An object type may participate in more than one role of a fact type, consider for example figure 4. In this figure, persons may participate either as parents, or as children (or both) in fact type *Parenthood.*

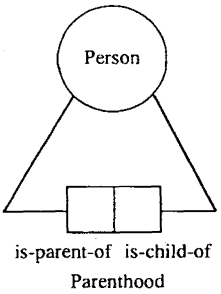


Figure 4. – A binary fact type

Fact types themselves may play roles in other fact types. Consider for example figure 5, where fact type *Enrollment* plays a role in another fact type. The schema of this figure models the fact that persons may achieve a score for a course in which they are enrolled. In this schema it is possible that a certain person has not (yet) achieved a score for a course in which (s)he is enrolled. In most conceptual data modeling techniques each instance of a fact type should have a value for *each* of the roles of that fact type, In other words, missing values in fact type instances do not occur. If for example the two fact types of the schema of figure 5 would be replaced by

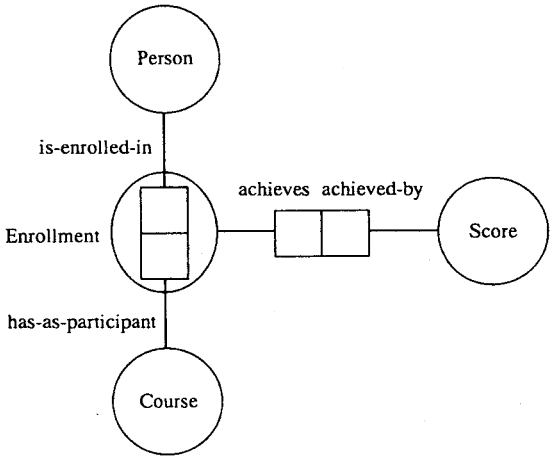


Figure 5. – An example of fact type participating in another fact type

a ternary fact type between *Person*, *Course*, and *Score*, each person has to have a score for each of the courses for which (s)he is enrolled.

2.3. Power types

While instances of fact types may be viewed as tuples, instances of *power types* may be viewed as sets. An instance of a power type can be seen as a (nonempty) set of instances of its *element type*, which has to be another object type of the data model. An instance of a power type is identified by its elements, just as a set is identified by its elements in set theory (axiom of extensionality). Hence, the instances of a power type are identifiable if and only if the elements of its element type are identifiable. Power typing corresponds to the notion of *grouping* as used in the IFO data model [2], the notion of *user-controllable grouping classes* in SDM [16] and the notion of *association* in [7].

A simple example of the application of power types can be found in the *Convoy Problem* [16], depicted in figure 6. In this diagram, the object type *Convoy* is a power type with element type *Ship*. As a result, each instance of object type *Convoy* can be considered as a set of instances of *Ship*. Convoys are identified by their constituent ships, whereas ships are identified by a *Ship-code*. In conceptual data modeling techniques without power types, the introduction of a *Convoy-code* would be required in order to identify the instances of object type *Convoy*, even when such a code is not present

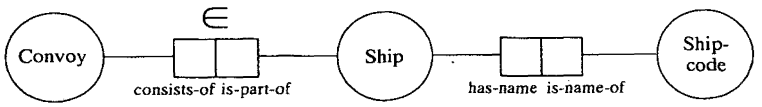


Figure 6. – A simple example of a power type

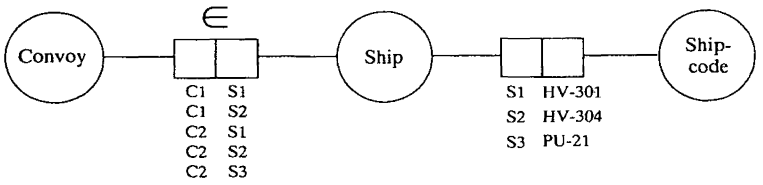


Figure 7. – A sample population of power type *Convoy*

in the problem domain. Furthermore, one explicitly needs to guarantee the extensionality property of sets.

As an example of a population of power type *Convoy*, consider figure 7. In this example there are two convoys, C_1 and C_2. Convoy C_1 can be viewed as the set $\{S_1, S_2\}$, while convoy C_2 can be viewed as the set $\{S_1, S_2, S_3\}$. Apparently, in this problem domain convoys may share ships.

In some data modeling techniques, instances of power types are simply sets, while in other data modeling techniques these instances have their own identity but exhibit set behavior (as in the population of figure 7). In this paper the latter approach is preferred as it is more general. From a dynamic point of view an important difference in semantics is that in this approach the identity of an instance of a power type remains unchanged even when the elements of that instance are changed.

Some data modeling techniques also offer other complex type constructors, such as *sequence types* allowing sequences of instances to be defined as instances. These type constructors are useful for system analysts but not elementary from a formal point of view, as they can be expressed in terms of other type construction mechanisms.

2.4. Specialization

Specialization is a mechanism for representing one or more (possibly overlapping) subtypes of an object type. Specialization is applied when certain facts are to be recorded only for specific instances of an object type. Suppose for example it is necessary to record cars owned by adults, *i.e.* persons with an age greater than or equal to 18. This situation is captured

by the schema in figure 8. Only instances of object type *Adult* can play the role *owns*.

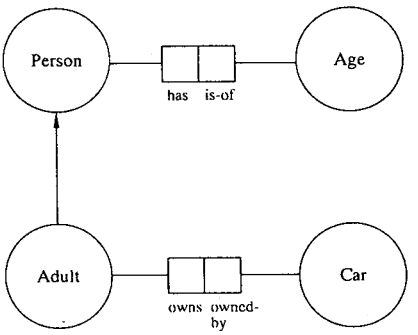


Figure 8. – A simple example of specialization

A specialization relation between a subtype and a supertype implies that the instances of the subtype are also instances of the supertype (each *Adult* is a *Person*). For proper specialization, it is required that subtypes are defined in terms of one or more of their supertypes. Such a decision criterion is referred to as a *subtype defining rule*. As such, specialization corresponds to the Comprehension Schema of formal set theory. The subtype defining rule then corresponds to the selection formula. Subtypes inherit the identification schemes of their supertypes. Therefore, if persons are identified by their name, adults are also identified by that name.

Figure 9 models a university using a schema that contains a subtype *network*. The department where they study is only recorded for students. The salary is only recorded for teachers, and the courses they teach are only recorded for teaching assistants. This example also demonstrates that *multiple inheritance* is in principle possible, since object type *Teaching-assistant* is a subtype of both *Teacher* and *Student*.

2.5. Generalization

Generalization is a mechanism that allows the population of a certain object type to be the union of the populations of other object types. Contrary to what its name suggests, generalization is *not* the inverse of specialization. Generalization originates from the Union Axiom of formal set theory. As generalization requires the covering of the generalized object type by its

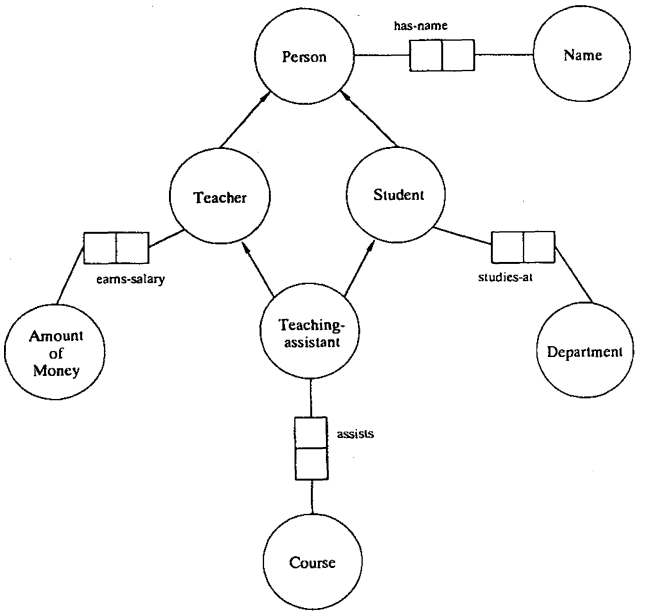


Figure 9. – Example of a specialization hierarchy

constituent object types (or *specifiers*), a decision criterion as in the case of specialization (the subtype defining rule) is not necessary.

As an example of the application of generalization, consider a problem domain where facts are to be recorded for cars and houses. When both for cars and houses a price is to be recorded, their respective object types *Car* and *House* may be generalized to an object type *Product*. This situation is captured by the schema of figure 10. The main difference with specialization is to be found in the inheritance of identification schemes. A generalized object type inherits the identification schemes from its specifiers. Hence, if in the example of figure 10, the object types *Car* and *House* are identifiable, the object type *Product* is also identifiable. Products are represented depending on their origin. In conceptual data modeling techniques without generalization (e.g. NIAM), this problem domain would require the use of specialization and consequently the (possibly artificial) introduction of a *Product-code* as subtypes have to inherit their identification from their supertypes.

Figure 11 contains a more complex example of generalization. A formula may be either a single variable, or constructed by some function (say f) from simpler formulas. This example demonstrates that generalization can be used to define recursive object types.

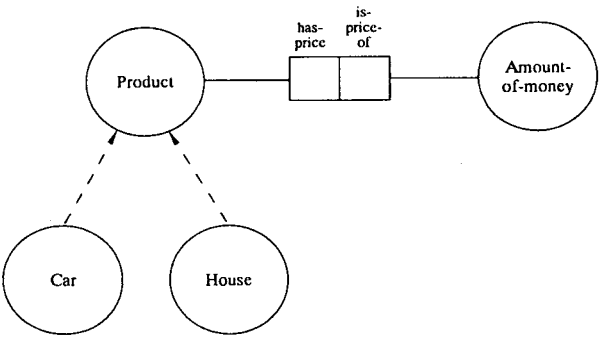


Figure 10. – Example of generalization

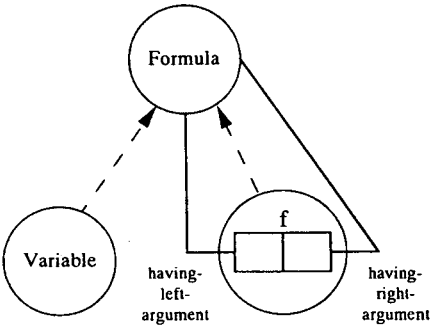


Figure 11. – Another example of generalization

2.6. Constraints

Constraints represent restrictions on populations. They exclude populations that do not correspond with a possible situation in the problem domain. Consider for example the schema of figure 4. In this schema it may be desirable to exclude cyclic parent-child relations. This implies the specification of a constraint that enforces the asymmetry of fact type *Parenthood.*

Two *types* of constraints frequently occur in conceptual data modeling: *total role* constraints and *uniqueness* constraints. Nearly every conceptual data modeling technique incorporates these types of constraints (sometimes in restricted variants). These types of constraints are important for the translation of conceptual models to implementation models. Total role constraints suggest mandatory fields, or combinations of fields, while uniqueness constraints can be used as a guarantee for integrity and as a base for efficient access mechanisms.

A total role constraint over a number of roles states that all instances in the object types playing these roles have to participate in at least one of these roles. In figure 6, the simplest example of a total role constraint would be a total role constraint over the role *has-name*. Such a total role constraint captures the requirement that each *Ship* has to have a *Ship-code*. In figure 4, a total role constraint over the roles *is-parent-of* and *is-child-of* expresses that each *Person* is either a parent or a child (or both). The general definition of a total role constraint allows the specification of such a constraint over roles associated with different object types.

A uniqueness constraint over a number of roles states that certain combinations of values in these roles should occur at most once. In its simplest form, *i.e.* all roles involved are part of the same fact type, a uniqueness constraint is also referred to as a *key*. A key on a number of roles of a fact type excludes that two different instances of that fact type have identical values in *all* these roles. A key on the role with name *has-name* in figure 6 expresses that each ship has at most one ship-code. A key on the role named *is-part-of* ensures that convoys do not share ships. More complex forms of uniqueness constraints are possible, e.g. over roles in different fact types, but not treated in the context of this paper.

Total role and uniqueness constraints play a crucial role in the determination of identification schemes. Total role constraints ensure that instances participate in the fact types used in the identification scheme, while uniqueness constraints ensure that no two different instances are related to exactly the same combination of values in those fact types. Consider for example the schema of figure 6. Ships may be identified by their code if and only if each ship has to have a ship-code (total role constraint on role *has-name*) and no two ships share the same code (uniqueness constraint on role *is-name-of*). If both conditions are fulfilled, ships may be denoted by their corresponding code without loss of information.

2.7. Type graphs

The focus in this section is on syntactical aspects of conceptual data models. A general syntactic description of a conceptual data model is a *type graph*. The nodes of the graph correspond to the object types and the labeled arrows determine the way they participate in the various constructions. The notion of a type graph presented in the following definition can be compared to the definitions presented in [28] and [32].

The set of nodes of a graph G is denoted as G_0 and its edges as G_1. The source of an edge e: $A \rightarrow B$ is denoted $\text{source}(e) = A$, and the target as $\text{target}(e) = B$.

DEFINITION 2.1: *A* type graph $\mathcal{G}$ *is a directed multigraph where the edges have labels from the set* $\{\text{role}, \text{spec}, \text{gen}, \text{power\_role}, \text{elt\_role}\}$, *such that there are no cycles consisting of edges with label* spec *or* gen. *Further there is a bijective function* pow *from edges with label* power_role *to edges with label* elt_role *such that* $\text{source}(\text{pow}(p)) = \text{source}(p)$. *The function* type *yields the label of an edge.* □

The relation between a power type and its element type is given its own identity. Roles are translated to edges labeled by role. The resulting graph is a *multi*graph, since an object type may participate via more than one role in a fact type.

The definition of a type graph is very liberal. The definition allows a node to be a power type as well as a fact type, a binary fact type to be a subtype of a ternary fact type, a power type to have several element types etc. Excluding these "peculiarities" from data models turns out to be unnecessary from a theoretical point of view as it is possible to give such data models a formal semantics. Hence, restrictions, other than on certain cycles, will not be imposed.

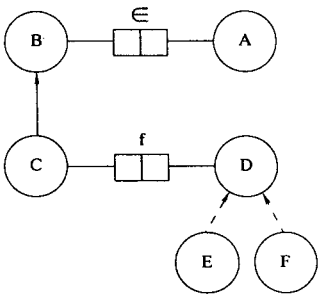


Figure 12. – A sample PSM schema

As an example of a type graph consider figure 13, which contains the associated type graph of the PSM schema of figure 12.

DEFINITION 2.2: *The set of* specifiers *of a node* n in the context of *a type graph* $\mathcal{G}$, *with nodes* $\mathcal{G}_0$ *and edges* $\mathcal{G}_1$, *is defined by:*

$$\{s \in \mathcal{G}_0 | \exists_{f:s \rightarrow n \in \mathcal{G}_1} [\text{type}(f) = \text{gen}]\}$$

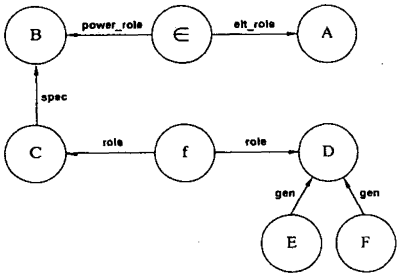


Figure 13. – Type graph of the schema of figure 12

The specifiers of node D in the context of the type graph of figure 13 are the nodes E and F.

The edges with type spec or gen are called the *subtype edges*. These edges play an important role in the category theory formalization of constraints as they identify corresponding instances. Subtype graphs do not contain edges other than subtype edges.

DEFINITION 2.3: *The* subtype graph *of a type graph $\mathcal{G}$ is defined as the subgraph of $\mathcal{G}$ with as nodes the nodes of $\mathcal{G}$,* i.e. *$\mathcal{G}_0$, and as edges, the subtype edges of $\mathcal{G}$.* □

The following definition is useful because the subtype relation should be transitive. A subtype path connects an object type with one of its supertypes.

DEFINITION 2.4: *A* subtype path *P in a type graph $\mathcal{G}$ is a path in the subtype graph of $\mathcal{G}$.* □

3. MATHEMATICAL PRELIMINARIES

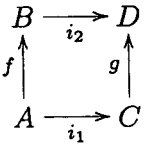
This section introduces the mathematical terminology and notations, that are used in this paper. In general these are similar to those in [10]. This section also contains some category theory proofs that are used later on.

3.1. Basic definitions

We start with some basic definitions and notations for some category theory properties. Isomorphisms are very important in category theory. The definition of isomorphic arrows is a natural one, although it is not used frequently.

DEFINITION 3.1: *An arrow* $f: A \rightarrow B$ *is an isomorphism iff an arrow* $g: B \rightarrow A$ *exists such that* $f \circ g = \mathrm{Id}_B$ *and* $g \circ f = \mathrm{Id}_A$. □

DEFINITION 3.2: *Two objects* A *and* B *are* isomorphic, *written* $A \cong B$, *iff an isomorphism* $f: A \rightarrow B$ *exists. Two arrows* $f: A \rightarrow B$ *and* $g: C \rightarrow D$ *are* isomorphic, *written* $f \cong g$, *iff isomorphisms* $i_1: A \rightarrow C$ *and* $i_2 : B \rightarrow D$ *exist such that the following diagram commutes:*

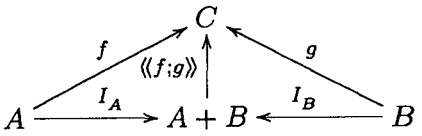


□

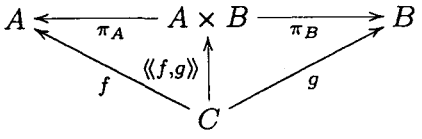
Next, notations for some limits and colimits are introduced.

The apex of a colimit cocone over a diagram D is denoted as γ_D, while the arrows of the cocone are denoted as $\alpha_D^n : n \rightarrow \gamma_D$ for a node n in D.

Given a coproduct $A + B$, and arrows $f : A \rightarrow C$ and $g : B \rightarrow C$ a unique arrow must exist, denoted $\langle\langle f; g\rangle\rangle : A + B \rightarrow C$, such that the following diagram commutes.



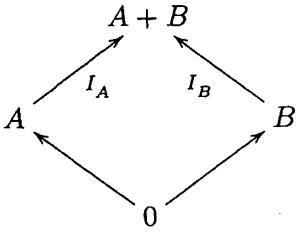
Likewise, given a product $A \times B$, and arrows $f : C \rightarrow A$ and $g : C \rightarrow B$ a unique arrow must exist, denoted $\langle\langle f, g\rangle\rangle : C \rightarrow A \times B$, such that the following diagram commutes.



3.2. Sums and complementability

In this paper sums are used frequently. As described in section 4, it is required that the instance category has disjoint sums.

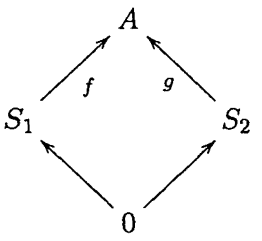
DEFINITION 3.3: *Let A and B be two objects in a category with an initial object 0 and a coproduct $A + B$. Then the following diagram commutes.*



If this diagram is a pullback and the canonical injections I_A and I_B are monomorphisms, then the coproduct $A + B$ is a disjoint coproduct. □

It is well known that coproducts in **Set** are disjoint. The coproduct of two sets A and B in **Set** is the amalgamated union that is different from the normal union, since common elements from A and B are represented twice. For example, the coproduct of $A = \{a, b\}$ and $B = \{a, c\}$ is the set $A + B = \{a', b', a'', c''\}$. The injection function from A to $A + B$ is $\{a \mapsto a', b \mapsto b'\}$ and the injection function from B to $A + B$ is $\{a \mapsto a'', c \mapsto c''\}$. It is shown in appendix A that several other interesting categories also have disjoint coproducts.

DEFINITION 3.4: *Let $f : S_1 \to A$ and $g : S_2 \to A$ be two monomorphisms, and let 0 be the initial object. If the following diagram is a pullback then the subobjects corresponding to f and g are* disjoint subobjects.



□

The notion of complemented subobject is the category theory generalization of complemented subsets in **Set**.

DEFINITION 3.5: *A morphism $f : A \rightarrow B$ is complementable iff there exists $g : C \rightarrow B$ such that $B \cong A + C$ with f and g as the coproduct arrows. In this case g is a* complement *of f. The object C is frequently notated as $B - A$.* □

Note that in a category in which all sums are disjoint all complementable morphisms must be monomorphisms.

DEFINITION 3.6: *A subobject corresponding to a monomorphism f is a* complementable subobject *iff f is complementable.* □

All monomorphisms in **Set** are complementable. Appendix A describes complementability in several other possible instance categories.

3.3. Properties of colimits

This section contains proofs of some properties of colimits that are used later on.

In our formalization subtype relations are represented by complementable monomorphisms. Further, it is necessary to compute colimits of diagrams that consist of complementable monomorphisms. The following lemmas are important for these computations.

LEMMA 3.1: *Given a category K in which all sums exist. Let $f : A \rightarrow B$ and $g : A \rightarrow C$ be morphisms in K such that f is complementable. Then the pushout P of f and g in K exists and the pushout morphism $p : C \rightarrow P$ is complementable.*

Proof: Since f is complementable there exists $\hat{f} : B - A \rightarrow B$.

Let $P = C + (B - A)$ with injection morphisms $I^P_C : C \rightarrow P$ and $I^P_{B-A} : B - A \rightarrow P$. Let $p = I^P_C$ and let $q : B \rightarrow P = \langle\langle p \circ g; I^P_{B-A}\rangle\rangle$. The following equations hold for q:

$$q \circ f = p \circ g \tag{1}$$

$$q \circ \hat{f} = I^P_{B-A} \tag{2}$$

We will now prove that P together with the arrows p and q forms a pushout of f and g. Let $r : B \rightarrow X$ and $s : C \rightarrow X$ be such that

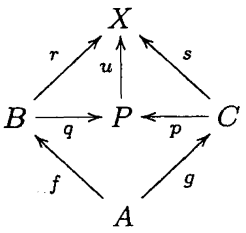
$$r \circ f = s \circ g \tag{3}$$

Then the universal pushout arrow u from P to X can be constructed by $u = \langle\langle r \circ \hat{f}; s\rangle\rangle$. u satisfies the following equations:

$$u \circ I^P_{B-A} = r \circ \hat{f} \tag{4}$$

$$u \circ p = s \tag{5}$$

If u is a valid pushout arrow the following diagram must commute:



Since we already have 1 and 5 it only remains to show that $u \circ q = r$. Let $v = \langle\langle r \circ f; r \circ \hat{f}\rangle\rangle$, then

$$v \circ f = r \circ f \tag{6}$$

$$v \circ \hat{f} = r \circ \hat{f} \tag{7}$$

Since v must be the unique solution to equations 6 and 7, it can be concluded that $v = r$. Using equations 6, 3, 5, and 1:

$$v \circ f = r \circ f = s \circ g = u \circ p \circ g = u \circ q \circ f \tag{8}$$

Using equations 7, 4, and 2:

$$v \circ \hat{f} = r \circ \hat{f} = u \circ I^P_{B-A} = u \circ q \circ \hat{f} \tag{9}$$

From equations 8 and 9 it can be seen that $v = u \circ q$ is also a solution to 6 and 7, and therefore $v = r = u \circ q$.

To complete the proof that P is a pushout it must still be shown that u is unique. Suppose there exists $u' : P \to X$ such that $u' \circ p = s$ and $u' \circ q = r$. From equation 2: $u' \circ I^P_{B-A} = u' \circ q \circ \hat{f} = r \circ \hat{f}$. Since u is the unique solution to equations 4 and 5 it can be concluded that $u = u'$.

From the construction process for P it is obvious that the pushout P can always be constructed if all sums exist. Furthermore, since p is a coproduct injection arrow it is by definition complementable. □

LEMMA 3.2: *Given a category K in which all sums exist and are disjoint. Let f: $A \rightarrow B$ and g: $A \rightarrow C$ be complementable monomorphisms in K. Then the pushout P of f and g in K exists and the pushout arrows are complementable monomorphisms.*

Proof: By applying the previous lemma twice it follows immediately that the pushout of f and g must exist and that the pushout arrows are complementable. Since all sums are disjoint in K all complementable arrows must be monomorphisms. □

LEMMA 3.3: *Given a category C that has an initial object and in which all sums exist and are disjoint. Let $D : G \rightarrow C$ be a diagram consisting of complementable monomorphisms. Then the colimit of D exists and its arrows are all complementable monomorphisms.*

Proof: All colimits can be constructed using the initial object and pushouts (e.g. *see* proposition 8.3.7 in [10]). The main proof goes by induction on this construction. The induction hypothesis is that all constructed arrows are complementable monomorphisms. In the construction process two cases can be distinguished:

• A pushout is constructed of two arrows from the initial object. Such a pushout is equivalent to a coproduct, as sums are disjoint the constructed arrows of the pushout must be complementable monomorphisms, because all sums exist this pushout also exists.

• A pushout is constructed of two arrows, that do not both have the initial object as source. For each such arrow there are two possibilities: the arrow is part of the diagram, in which case it must be a complementable monomorphism, or the arrow was constructed in a previous step, and by our induction hypothesis it must also be a complementable monomorphism. Application of lemma 3.2 yields that the pushout exists and that constructed arrows are complementable monomorphisms. □

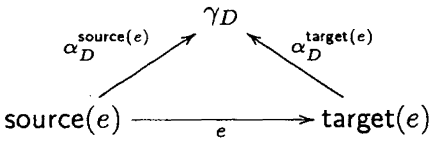
The following two lemmas describe how the colimit of a diagram changes if the diagram is extended.

LEMMA 3.4: *Given a commutative diagram D such that the colimit of D exists. Let D' be the diagram that consists of D extended with the arrows and apex of its colimit. Then diagram D' commutes.*

Proof: Let P_1 and P_2 be two paths in D' that both start in the same object S and both end in object T. Since there are no arrows leaving γ_D, it must always be the last element in any path that contains γ_D. So, two cases can be distinguished:

• $T \neq \gamma_D$. In this case both P_1 and P_2 contain edges from D only and therefore they commute.

• $T = \gamma_D$. Let L_1 be the last but one object in P_1 and L_2 be the last but one object in P_2. The last edge in P_1 must be $\alpha_D^{L_1}$ and the last edge in P_2 must be $\alpha_D^{L_2}$. All other edges in both paths must be part of D. From the definition of the colimit, for every edge e in D the following diagram commutes:

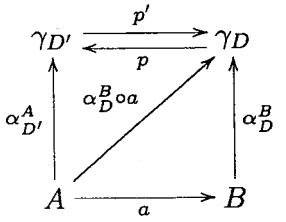


Thus the subpath $\alpha_D^{\text{target}(e)} \circ e$ can be replaced by $\alpha_D^{\text{source}(e)}$. We can prove by induction on the path that P_1 and P_2 must both be equal to α_D^S. Therefore the entire diagram D' commutes. □

LEMMA 3.5: *Given a diagram D and an arrow $a : A \to B$ where A is a node that is not in D, and B is a node that is in D. Let D' be D extended by a. Then the apex of the colimit of D is isomorphic to the apex of the colimit of D', provided that both colimits exist.*

Proof: Omitting $\alpha_{D'}^A$ from the colimit of D' gives a cocone over D. Thus a unique arrow $p : \gamma_D \to \gamma_{D'}$ exists. In a similar way, adding the arrow $\alpha_D^B \circ a$ to the colimit of D gives a cocone over D'. This implies the existence of a unique arrow $p' : \gamma_{D'} \to \gamma_D$. Due to the *couniversal* properties of the colimit

$p' \circ p = \mathrm{Id}_{\gamma_D}$ and $p \circ p' = \mathrm{Id}_{\gamma_{D'}}$. Therefore, γ_D and $\gamma_{D'}$ are isomorphic.



□

4. THE TYPE MODEL

Having introduced the syntax of type graphs, we will now describe their semantics. As is customary, the semantics is described as a set of models. A type model captures the notion of a population of a given type graph. A population corresponds to a state of the problem domain. In this section only type graphs without constraints are considered, the semantics of constraints are described in the following section.

DEFINITION 4.1: *A* model *M of a graph G in a category C is a graph homomorphism from G to C.* □

DEFINITION 4.2: *Given a category F in which all sums and products exist and in which all sums are disjoint, a* type model *for a given type graph $\mathcal{G}$ in F, is a model $M : \mathcal{G} \rightarrow F$. F is referred to as the* instance category *of the model.* □

The type model maps the object types in a type graph onto objects in the instance category and the edges onto arrows in this category. To avoid notational clutter the model homomorphism is sometimes omitted if it is clear from the context. For example, the product of two object types is sometimes written as $A \times B$ instead of $M(A) \times M(B)$.

This type model has to satisfy several requirements, that are introduced in the following paragraphs. The full definition of a *valid* type model for a type graph is presented in section 4.5.

4.1. Fact types

In the past, fact types have often been formalized by viewing them as subsets of a cartesian product. This has commonly been referred to as the

tuple oriented approach. As an example consider figure 1. A population of this fact type, represented in the tuple oriented approach, could be:

$$\mathrm{Pop}(R) = \{\langle a_1, b_1\rangle, \langle a_2, b_1\rangle\}.$$

The disadvantages of the tuple oriented approach are obvious: the representation of instances is overly specific. Instances of fact type R could as well be considered elements of the product $\mathrm{Pop}(B) \times \mathrm{Pop}(A)$ as $\mathrm{Pop}(A) \times \mathrm{Pop}(B)$. A cartesian product imposes an ordering on the various parts of the relation. Consequently, algebraic operators do not have important properties such as commutativity and associativity. This observation has led to the *mapping oriented approach* [22], where fact type instances are treated as functions from the involved roles to values. In this approach, the above sample population would be represented as:

$$\mathrm{Pop}(R) = \{\{r \mapsto a_1, s \mapsto b_1\}, \{r \mapsto a_2, s \mapsto b_1\}\}.$$

Clearly, this approach does not suffer from the drawbacks of the tuple oriented approach. No ordering is imposed, while at the same time the various parts of a relation remain distinguishable.

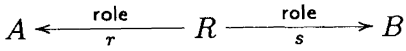


Figure 14. – Type graph of the schema of figure 1

Still, however, one may argue that the mapping oriented approach imposes unnecessary restrictions. Why do instances have to be represented as *functions*? Isn't it sufficient to have access to their various parts? The categorical approach pursues this line of thought. The actual representation of fact type instances becomes irrelevant, their components become available by "access-functions". As an example consider the interpretation of the sample population in the category **FinSet**. The type graph of the schema of figure 1 is shown in figure 14. Category theoretically, a population corresponds to a mapping from the type graph to an instance category. The sample population therefore, could be represented as (note that there are many alternatives!):

$$r = \{f_1 \mapsto a_1, f_2 \mapsto a_2\},$$
$$s = \{f_1 \mapsto b_1, f_2 \mapsto b_1\}.$$

In this approach, the two fact type instances have an identity of their own, and the functions r and s can be applied to retrieve the respective components.

Note that in this approach it is possible that two different fact type instances consist of exactly the same components. This is in line with object oriented approaches, where different objects may have identical properties.

Apart from **FinSet** it is also possible to choose other instance categories, a subject further elaborated upon in section 6.1. The category **PartSet**, where the objects are sets and the morphisms partial functions, allows certain components of fact type instances to be undefined:

$$r = \{f_2 \mapsto a_2\},$$
$$s = \{f_1 \mapsto b_1, f_2 \mapsto b_1\}.$$

In this population, fact type instance f_1 does not have a corresponding object playing role r. Clearly, **PartSet** can be used to model techniques where missing (or unknown) values are allowed.

Another possible choice of instance category is the category **Rel**, where the objects are sets and the morphisms relations. In **Rel** the components of fact type instances correspond to sets, as roles are mapped on relations. A fact type instance may be related to one or more objects in one of its components. A sample population could be:

$$r = \{f_2 \mapsto a_1, f_2 \mapsto a_2\},$$
$$s = \{f_1 \mapsto b_1, f_2 \mapsto b_1, f_2 \mapsto b_2\}.$$

As can be seen, the categorical formalization of fact types does not impose any extra requirements on the definition of type model.

4.2. Subtype relationships

The model must express the subtype relationships between different object types. The subtype diagram describes these relationships between object types.

DEFINITION 4.3: *The* subtype diagram *of a model M of a graph G with subtype graph S is M functionally restricted to S.* □

The arrows in the subtype diagram represent the injections from an object type to one of its supertypes.

Intuitively, all elements in an object type must be uniquely represented in all of their supertypes. In other words, we want (categorically) that each subobject of an object type corresponds with a unique subobject of each of its supertypes.

As multiple inheritance is allowed it is possible that there are several different paths in the subtype graphs between two object types. The result of injecting an object type into one of its supertypes should not depend on which of these paths is selected. Therefore, we require that all paths in the subtype diagram between two nodes should have the same value in the type model. To put it differently, the subtype diagram must commute.

Intuitively, each instance of an object type should correspond to a unique instance in each of its supertypes. Translated into category theory terms this becomes: each subobject in a given object type should correspond with a unique subobject in each of its supertypes. As the following lemma shows, a sufficient condition for this requirement is that all arrows in the subtype diagram are monomorphisms.

LEMMA 4.1: *Let T and S be object types such that a path P exists in the subtype graph between T and S. If all arrows in the subtype diagram are monomorphisms then each subobject of $M(T)$ corresponds to a unique subobject of $M(S)$.*

Proof: Let a_1, a_2 be two arrows that are elements of the same subobject of $M(T)$. The definition of a subobject implies that a_1 and a_2 must factor one another. The composition of two monomorphisms is also a monomorphism. Therefore, $M(P) \circ a_1$ and $M(P) \circ a_2$ are also monomorphisms that factor one another and are therefore element of the same subobject of S.

If we consider two arrows a_1, a_2 that are elements of different subobjects of $M(T)$, then $M(P) \circ a_1$ and $M(P) \circ a_2$ must be elements of different subobjects of $M(S)$. If this were not the case $M(P) \circ a_1$ and $M(P) \circ a_2$ would factor through one another, and because $M(P)$ is a monomorphism this would imply that a_1 and a_2 would also factor through one another, which is contrary to the assumption that they are not part of the same subobject of T. □

4.3. Generalization

A generalized object type is the union of a given set of object types, called its *specifiers*. Simply using sums for generalized types does not work, because sums are disjoint in the instance category. This implies that instances of object types that are a subtype of more than one specifier would be represented multiple times in the sum. This problem can be solved by using a more general colimit.

The collection of instances of a generalized type with a set of specifiers V is completely determined by the subtype relationships among the subtypes of elements in V. The following definitions give a formal description of a diagram that only contains the relevant subtype relations among subtypes of elements of V.

DEFINITION 4.4: *Given a graph G and a set of nodes $N \subseteq G_0$, G* dominated by *N is equal to a subgraph D of G that is defined as follows: The edges of D are the edges from G_1 that occur on a directed path that ends in a node $n \in N$. The nodes of D are the nodes that occur in one of its edges.* □

DEFINITION 4.5: *Given a diagram $D : G \to C$ and a set of nodes $V \subseteq G_0$. Let G_V be G dominated by V. Then, D* dominated by *V is equal to D functionally restricted to G_V.* □

The instance universe U_M^V represents the collection of all instances of a set V of object types. The instance universe is used as the generalization of a set V of specifiers.

DEFINITION 4.6: *The* instance universe *determined by a set of object types $V \subseteq \mathcal{G}_0$ in a given type model M, denoted as U_M^V, is the apex of the universal cocone with as base the subtype diagram dominated by V.* □

The remainder of this section contains some lemmas that prove that the use of the instance universe for generalization corresponds with the intuitive properties that generalization should have.

In many practical data models the subtype graph forms a forest. If the subtype graph dominated by V forms a forest, different roots should not have instances in common. Therefore, the generalization of V should be equal to the disjoint sums of the roots.

LEMMA 4.2: *If the subtype graph dominated by V is a forest then U_M^V is equal to the sum of the roots of the trees in the forest.*

Proof: From lemma 3.5 it follows that the apex of the colimit of a forest-like diagram is equal to the colimit of this diagram after removing one of the leaves. In this way all nodes in the diagram except the roots of the trees can be removed. This leaves a discrete diagram and the colimit of such a diagram is the sum of the nodes. □

As explained in section 4.2, it is desirable that the subtype diagram commutes and that its arrows are all monomorphisms. Further, the gene-

ralization of a set of specifiers V should always be computable, in other words the colimit U_M^V must exist. A sufficient condition for having generalization satisfy all of these requirements is that all subtype arrows are *complementable* monomorphisms. The following lemma shows that these requirements are indeed satisfied if subtype arrows are complementable.

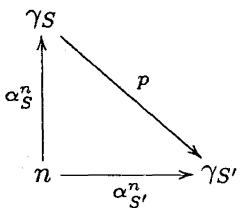
LEMMA 4.3: *Let C be a category in which all sums exist and are disjoint. Let S be a commuting diagram in C whose arrows are complementable monomorphisms. Then for any set V of objects in S it is possible to extend S with an object type that is a generalization of V. The resulting diagram commutes and its arrows are complementable monomorphisms.*

Proof: Since all sums in C are disjoint it follows immediately from lemma 3.3 that U_M^V exists, and that all newly constructed arrows are complementable monomorphisms. Since S is a commuting diagram it follows from lemma 3.4 that the newly constructed diagram must also commute. □

Generalization should not create any new object type instances. The following lemma shows that this is indeed the case.

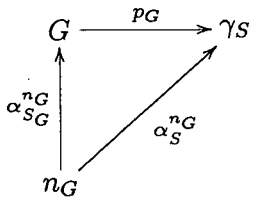
LEMMA 4.4: *Let S be a subtype diagram and let O be a set of objects in S. Let S' be the diagram that is constructed by adding a new object type G that is the generalization of O, together with the corresponding generalization arrows from the elements of O to G. The apex of the colimit of S is isomorphic to the apex of the colimit of S'.*

Proof: Removing $\alpha_{S'}^G$ from the colimit of S' leaves a cocone over S. Therefore, a unique arrow $p : \gamma_S \to \gamma_{S'}$ exists such that for all nodes n in S the following diagram commutes:

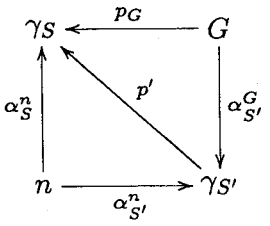


Let S_G be equal to S dominated by O. From the definition of a generalized object type we have $G = \gamma_{S_G}$. As S_G is a subdiagram of S there is a unique arrow $p_G : G \to \gamma_S$, such that for all nodes n_G in S_G the following diagram

commutes:



Observe that S' is equal to S plus $\{\alpha^o_{S_G} | o \in O\}$. After adding p_G to the colimit of S a cocone over S' is obtained. Therefore a unique arrow $p' : \gamma_{S'} \rightarrow \gamma_S$ must exist, such that for all nodes n in S' the following diagram commutes:

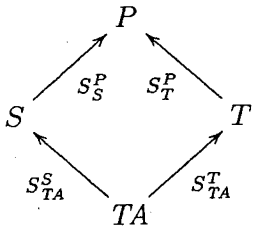


Due to the universal properties of the colimit $p' \circ p = \mathrm{Id}_{\gamma_S}$ and $p \circ p' = \mathrm{Id}_{\gamma_{S'}}$. γ_S is therefore isomorphic with $\gamma_{S'}$. □

It is possible to have generalizations of generalized types. Since the subtype graph may not contain cycles, such generalizations can always be computed. It would have been possible to allow cycles, but then the definition of instance universe would be more complicated. Further, this extra generality is not very useful because all object types that are part of a cycle in the subtype graph must be isomorphic.

Example 4.1: The use of generalizations will be illustrated with an example that is loosely based on figure 9. The example describes a university with students (S) and teachers (T). Teaching assistants (TA) are a specialization of both types. The object type person P is a generalization of students and

teachers. So the model gives the following diagram, that is a pushout.



Since the subtype diagram must commute, we have that

$$S_S^P \circ S_{TA}^S = S_T^P \circ S_{TA}^T,$$

which means that every teaching assistant has a unique representation as a person. As P is a generalization of S and T we have that

$$P = (S - TA) + TA + (T - TA).$$

□

4.4. Power types

Power types are object types whose elements are sets of elements of other object types. Each power type has a relation that relates the elements of the power type to the elements of the underlying object type. An essential feature of a power type is that its elements, that are essentially sets, are identified by their members in the underlying object type. This means that there can be no two elements of a power type with the same set of members. In other words, it is not possible to interchange two different elements of a power type without structurally altering the corresponding element-of relation.

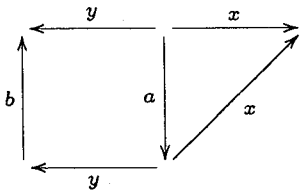
Suppose there is a power type P with corresponding underlying object type E and power-relationship object type R, with the corresponding edges $e : R \to E$ and $p : R \to P$ with $\text{type}(e) = \text{elt\_role}$ and $\text{type}(p) = \text{power\_role}$ such that $\text{pow}(p) = e$.

If two elements of the power type have the same elements then a permutation of the power type exists that leaves the member relationship essentially unchanged. In "set-like" categories permutations are equivalent to automorphisms, *i.e.* isomorphisms that are endomorphisms. So a first intuitive guess would be to require that no automorphism $a : M(P) \to M(P)$ with $a \neq \text{Id}_{M(P)}$ such that $a \circ M(p) = M(p)$ exists, however, this is not sufficient as the following example shows.

Example 4.2: This example uses a model $M : \mathcal{G} \rightarrow \mathbf{Set}$. $M(E) = \{e_1\}$, $M(P) = \{p_1, p_2\}$, $M(R) = \{r_1, r_2\}$, $M(e) = \{r_1 \mapsto e_1, r_2 \mapsto e_1\}$, $M(p) = \{r_1 \mapsto p_1, r_2 \mapsto p_2\}$. Essentially this describes two sets p_1 and p_2 that have the same member e_1. If we choose the obvious automorphism $a = \{p_1 \mapsto p_2, p_2 \mapsto p_1\}$ it is not true that $a \circ M(p) = M(p)$. □

The problem here is caused by the fact that elements of $M(R)$ have their own identity. We are not interested in the identities of the elements in $M(R)$. These elements (*i.e.* the tuples in the relationship) can be permuted in arbitrary ways without affecting the power sets. Thus if there is an automorphism $b : R \rightarrow R$ then $e \circ b$ and $p \circ b$ are for our purposes equivalent to e and p respectively. This leads to the following definition of the extensionality property.

DEFINITION 4.7: *Two arrows x and y, with* $\text{source}(x) = \text{source}(y)$, *fulfill the* extensionality *property iff, for all automorphisms* $b : \text{target}(y) \rightarrow \text{target}(y)$ *such that* $b \neq \text{Id}_{\text{target}(y)}$, *no automorphism* $a : \text{source}(y) \rightarrow \text{source}(y)$ *exists such that* $b \circ y \circ a = y$ *and* $x \circ a = x$



□

The arrows from the previous example do not have the extensionality property, as can be seen by selecting $b = \{r_1 \mapsto r_2, r_2 \mapsto r_1\}$.

4.5. Definition of valid type model

We now present the full definition of a valid type model for a type graph.

DEFINITION 4.8: *A type model $M : \mathcal{G} \rightarrow F$ for a given type graph $\mathcal{G}$ in a category F, is a* valid type model *iff,*

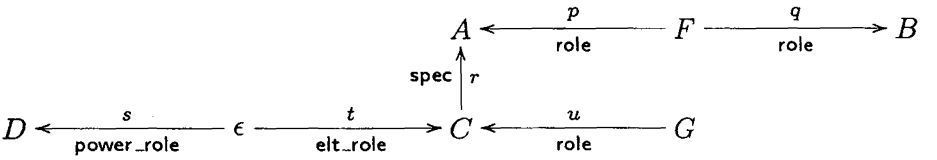
1. *if x is an edge of $\mathcal{G}$ and* $\text{type}(x) = \text{spec}$ *then $M(x)$ is a complementable monomorphism.*

2. *if x is an edge of $\mathcal{G}$ and* $\text{type}(x) = \text{gen}$ *then* $M(x) = \alpha_D^{\text{source}(x)}$, *where D is equal to the subtype diagram dominated by the specifiers of* $\text{target}(x)$.

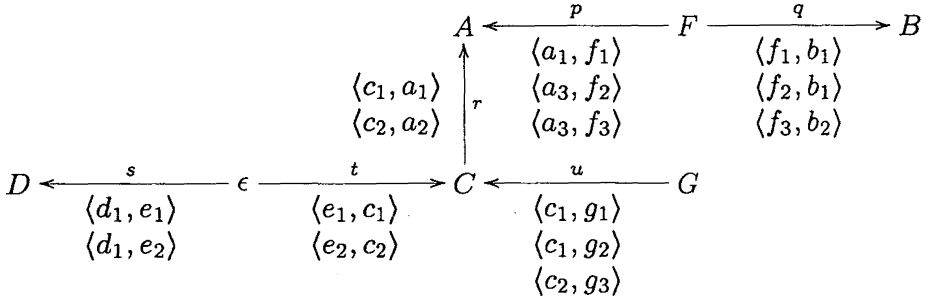
3. *the subtype diagram of M commutes.*

4. *if x and y are edges of $\mathcal{G}$, with* $\mathrm{pow}(y) = x$ *then $M(x)$ and $M(y)$ have to fulfill the extensionality property.* □

Example 4.3: The following type graph describes a simple conceptual data model.



The following is a type model of this type graph in **Set**. The value of the set of elements for each object type is equal to the elements that occur in the corresponding arrows, and has therefore been omitted from the figure.



This type model is indeed a valid type model. There is one specialization arrow from C to A that is an injective function, and in **Set** all injective functions are complementable monomorphisms. Obviously, the subtype diagram commutes since it only contains one specialization arrow. The power type D has one instance that represents the set $\{c_1, c_2\}$. It is not difficult to see that s and t fulfill the extensionality property. □

5. CONSTRAINTS

Each schema can have several associated constraints. This section describes the semantics of constraints. A constraint puts further restrictions on the set

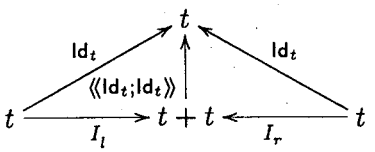
of valid type models for a given schema. Two important kinds of constraints that are used frequently in conceptual data modeling techniques are the total role constraint and the uniqueness constraint. The semantics of these constraints are described in the following sections. The basic framework described in this paper could be extended by arbitrary constraints that can be described in a category theory way.

5.1. Total role constraint

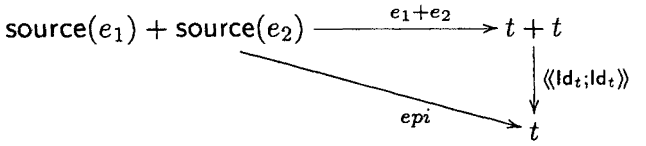
The intuitive semantics of a total role constraint is that all elements of a given set of object types participate in a given set of roles. A total role constraint is determined by a set of edges $\tau \subseteq \mathcal{G}_1$.

In the simplest example of a total role constraint, τ consists of a single edge e. This total role constraint means that all elements of $\mathrm{target}(e)$ must participate in e. In a model in the category **Set** this implies that $M(e)$ must be a surjective function. More generally we require that $M(e)$ must be an epimorphism.

A slightly more complicated example is $\tau = \{e_1, e_2\}$. Two cases can be distinguished, depending on whether both edges have the same target. In the first case both arrows have the same target $t = \mathrm{target}(e_1) = \mathrm{target}(e_2)$. The intuitive meaning of this constraint is that each element of t must participate in at least one of these two edges. For the semantics of the constraint, first construct the sum arrow $e_1 + e_2 : \mathrm{source}(e_1) + \mathrm{source}(e_2) \rightarrow t + t$. Intuitively speaking each element of t must be present in $\mathrm{target}(e_1 + e_2)$, however, as $t + t$ is a disjoint sum every element is represented twice. Therefore an arrow is needed that maps each element of $t + t$ onto the corresponding element of t. This can be achieved as follows. From the definition of the coproduct it follows that there are two injection arrows $I_l : t \rightarrow t + t$ and $I_r : t \rightarrow t + t$. Further, there is a unique arrow $\langle\langle \mathrm{Id}_t; \mathrm{Id}_t \rangle\rangle : t + t \rightarrow t$, such that the following diagram commutes.



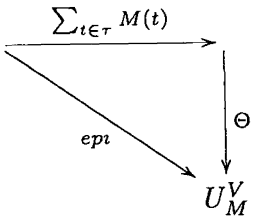
The meaning of the total role constraint is that $\langle\langle \mathrm{Id}_t; \mathrm{Id}_t\rangle\rangle \circ (e_1 + e_2)$ must be an epimorphism.



If $\mathrm{target}(e_1) \neq \mathrm{target}(e_2)$, it is possible that one of these is a subtype of the other. In this case we first inject the elements of the subtype into the supertype and then follow the same procedure as in the previous case. Note that the supertype is always equal to $U_M^{\{\mathrm{target}(e_1),\mathrm{target}(e_2)\}}$.

The full definition of the semantics of the total role constraint is given below.

DEFINITION 5.1: *Given a valid type model M and a total role constraint over $\tau \subseteq \mathcal{G}_1$. Let $s = \sum_{t\in\tau} M(t)$, $V = \{\mathrm{target}(M(t))|t \in \tau\}$. The definition of the instance universe U_M^V implies that for each $t \in \tau$ an arrow $i_t : \mathrm{target}(M(t)) \to U_M^V$ exists. Since $\mathrm{target}(s)$ is a coproduct, these i_t determine a unique arrow $\Theta : \mathrm{target}(s) \to U_M^V$.* M satisfies the total role constraint τ *iff $\Theta \circ s$ is an epimorphism.*



□

The total role constraint can be seen as a generalization of several types of constraints found in other data modeling techniques, such as the collection cover constraint and the subtype cover constraint. The collection cover constraint for a power type specifies that all instances of its element type should participate in at least one of its instances. The subtype cover constraint specifies that all instances of a given object type should be instances of at least one of a given set of subtypes of that object type.

Example 5.1: In example 4.3 take the total role constraint over $\tau = \{p, u\}$. Then $V = \{A, C\}$ and $U_M^V = A$. The sum $p + u : F + G \to A + C$ is the function $\{f_1 \mapsto a_1, f_2 \mapsto a_3, f_3 \mapsto a_3, g_1 \mapsto c_1, g_2 \mapsto c_1, g_3 \mapsto c_2\}$. Then $\Theta : A + C \to A = \{a_1 \mapsto a_1, a_2 \mapsto a_2, a_3 \mapsto a_3, c_1 \mapsto a_1, c_2 \mapsto a_2\}$. The composition $\Theta \circ (p + u) = \{f_1 \mapsto a_1, f_2 \mapsto a_3, f_3 \mapsto a_3, g_1 \mapsto a_1, g_2 \mapsto a_1, g_3 \mapsto a_2\}$ is an epimorphism in **Set** because it is a surjective function. Therefore, the total role constraint over $\tau = \{p, u\}$ is satisfied in this model.

The total role constraint over $\{p\}$ is not satisfied, but the total role constraint over $\{q\}$ is satisfied in this model. □

5.2. Uniqueness constraint

The uniqueness constraint is closely related to the concept of a key over a relation. A uniqueness constraint is determined by a set of edges $\tau \subseteq \mathcal{G}_1$.

In the most trivial case τ consists of a single edge e. The intuitive semantics is that each element of $\text{target}(e)$ determines at most one element in $\text{source}(e)$. For a model M in the category **Set** this implies that $M(e)$ must be an injective function. More generally, $M(e)$ must be a monomorphism.

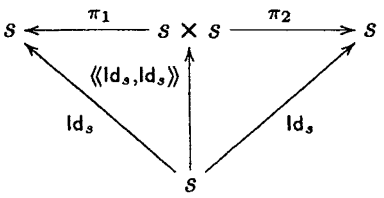
In the next and more interesting case $\tau = \{e_1, e_2\}$ with

$$\text{source}(e_1) = \text{source}(e_2) = s.$$

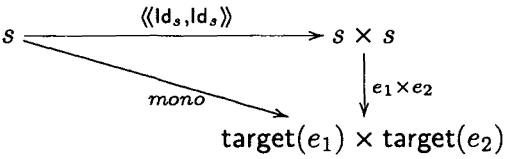
In this case the intuitive semantics is that the combination of an element from $\text{target}(e_1)$ with an element from $\text{target}(e_2)$ determines at most one element in $\text{source}(e_1)$. Start by constructing the product arrow

$$e_1 \times e_2 : s \times s \to \text{target}(e_1) \times \text{target}(e_2).$$

From the definition of the product it follows that there are two projection arrows $\pi_1 : s \times s \to s$ and $\pi_2 : s \times s \to s$. Further, there is a unique arrow $\langle\langle \text{Id}_s, \text{Id}_s \rangle\rangle : s \to s \times s$, such that the following diagram commutes.



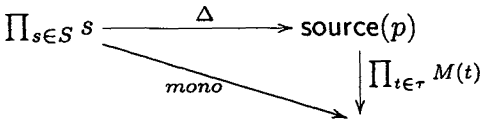
The meaning of the uniqueness constraint is that $(e_1 \times e_2) \circ \langle\langle \mathrm{Id}_s, \mathrm{Id}_s \rangle\rangle$ must be a monomorphism.



The case that $\tau = \{e_1, e_2\}$ with $\mathrm{source}(e_1) \neq \mathrm{source}(e_2)$ is simple, because it is equivalent to the combination of two uniqueness constraints, one over $\{e_1\}$ and the other over $\{e_2\}$.

The full definition of the semantics of the uniqueness constraint is given below.

DEFINITION 5.2: *Given a valid type model M and a uniqueness constraint over $\tau \subseteq \mathcal{G}_1$. Let $p = \prod_{t \in \tau} M(t)$, $S = \{\mathrm{source}(M(t)) | t \in \tau\}$. For each $t \in \tau$ there is an arrow $\pi_t : \prod_{s \in S} s \rightarrow \mathrm{source}(M(t))$. From the definition of the product it follows that these π_t determine a unique arrow $\Delta : \prod_{s \in S} s \rightarrow \mathrm{source}(p)$. Then, M satisfies the uniqueness constraint τ iff $p \circ \Delta$ is a monomorphism.*



□

Fact types behave by default as multisets: the same tuple can be represented multiple times. If this is undesirable, it can be avoided by adding a uniqueness constraint over the roles of the fact type.

Example 5.2: Take for instance, in example 4.3, the uniqueness constraint over $\tau = \{p, q\}$. Intuitively speaking, this constraint should be satisfied since every combination from A and B determines at most one element of F. The arrow $\Delta : F \rightarrow F \times F = \{f_1 \mapsto \langle f_1, f_1 \rangle, f_2 \mapsto \langle f_2, f_2 \rangle, f_3 \mapsto \langle f_3, f_3 \rangle\}$. The product $p \times q : F \times F \rightarrow A \times B =$

$$\{\langle f_1, f_1 \rangle \mapsto \langle a_1, b_1 \rangle,\ \langle f_1, f_2 \rangle \mapsto \langle a_1, b_1 \rangle,\ \langle f_1, f_3 \rangle \mapsto \langle a_1, b_2 \rangle,$$
$$\langle f_2, f_1 \rangle \mapsto \langle a_3, b_1 \rangle,\ \langle f_2, f_2 \rangle \mapsto \langle a_3, b_1 \rangle,\ \langle f_2, f_3 \rangle \mapsto \langle a_3, b_2 \rangle,$$
$$\langle f_3, f_1 \rangle \mapsto \langle a_3, b_1 \rangle,\ \langle f_3, f_2 \rangle \mapsto \langle a_3, b_1 \rangle,\ \langle f_3, f_3 \rangle \mapsto \langle a_3, b_2 \rangle\}$$

The composition

$$(p \times q) \circ \Delta = \{f_1 \mapsto \langle a_1, b_1 \rangle, f_2 \mapsto \langle a_3, b_1 \rangle, f_3 \mapsto \langle a_3, b_2 \rangle\}$$

is a monomorphism, because it is an injective function. Therefore, the uniqueness constraint over $\tau = \{p, q\}$ is satisfied.

The separate uniqueness constraints over $\{p\}$ and $\{q\}$ are not satisfied because p and q are not monomorphisms. □

6. INSTANCE CATEGORIES

One of the most important advantages of using a category theory approach to the semantics of conceptual data modeling techniques is that different instance categories can be used. The requirements that instance categories should satisfy are listed together with some illustrations.

Instance categories should support the constructions that have been used in the previous sections. This means that an instance category should have the following properties:

- All finite sums and products must exist.
- Sums must be disjoint.
- An initial object must exist.

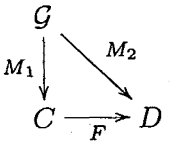
Actually, the last requirement is redundant since the initial object is the sum of zero objects. This set of requirements is very modest, which implies that there is a large set of possible instance categories.

As noted in section 5 the framework that is described in this paper could be expanded by adding new types of constraints. Of course the constructions that are needed by such constraints could further restrict the set of instance categories.

Some categories, however, are too trivial to be interesting as instance categories, for example the category with only one object and one arrow. The "standard" instance category that has been used for other formalizations of conceptual data modeling techniques is the category **FinSet**. It seems therefore reasonable to require that other instance categories have at least the same "expressive power". Intuitively, each model in **FinSet** should have a counterpart in other instance categories.

As an introduction to the formalization of this requirement it is useful to define a homomorphism between type models.

DEFINITION 6.1: *A* type model homomorphism *between type models* $M_1 : \mathcal{G} \rightarrow C$ *and* $M_2 : \mathcal{G} \rightarrow D$ *is a functor* $F : C \rightarrow D$ *such that the following diagram commutes:*



□

The valid type models and their homomorphisms form a category.

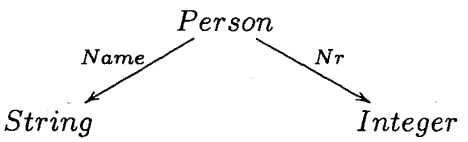
This definition of a type model homomorphism has inspired the following definition of a valid instance category.

DEFINITION 6.2: *A category* C *is a* valid instance category *iff a functor* $F : \mathbf{FinSet} \rightarrow C$ *exists such that for all* $M_1, M_2 : \mathcal{G} \rightarrow \mathbf{FinSet}$ *it holds that* $M_1 = M_2 \Leftrightarrow F \circ M_1 = F \circ M_2$. □

6.1. Sample instance categories

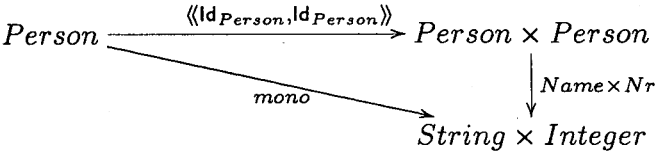
The following categories are valid instance categories: **FinSet**, **Set**, **PartSet**, **Rel**, **FuzzySet**. A description of various category theory constructs and proofs for these categories can be found in appendix A.

Models in **PartSet** give a way to handle missing values. Suppose that persons are identified by their names. Two different persons with identical names get an additional number to distinguish them.



The arrow Nr is a partial function, because persons with a unique name do not have a number. Suppose that we want to express that every person

must be uniquely identified by a combination of name and number. This can be achieved by putting a uniqueness constraint over $\{Name, Nr\}$.



The arrow $\langle\langle \mathrm{Id}_{Person}, \mathrm{Id}_{Person}\rangle\rangle$ is equal to $\{p \mapsto \langle p, p\rangle | p \in Person\}$. The arrow $Name \times Nr$ is interesting, since it maps the tuple $\langle p, p\rangle$ for a person p whose Nr is undefined to the tuple $\langle Name(p), \perp\rangle$, as described in appendix A. The uniqueness constraint holds if

$$(Name \times Nr) \circ \langle\langle \mathrm{Id}_{Person}, \mathrm{Id}_{Person}\rangle\rangle$$

is a monomorphism, *i.e.* a total injective function. This implies that two persons with the same name must have different numbers, which was indeed the requirement we tried to express.

In several object-oriented databases [21], [34], objects can have multi-valued (or set-valued) attributes. This means that the value of an attribute can be a (possibly empty) set of attribute values. Models in the category **Rel** can be used to model this behavior.

Models in **FuzzySet** can be used to model uncertainties. Every object type A is equipped with a function σ_A that describes the probability that an element is a member of that object type. The arrows in **FuzzySet** are total functions, and for each arrow $f : A \rightarrow B$ it must hold that $\sigma_A(a) \leq \sigma_B(f(a))$. Therefore, the probability that an individual is an element of a given object type must always be $\geq$ the probability that this individual is an element of one of the subtypes of this object type. Intuitively, this is sensible since if the individual is an element of an object type it must certainly be an element of all supertypes of that type. If we look at the simple example for a fact type that was shown in figure 2, we find that the probability that a given person is enrolled in a given course must be less than the probability that that person exists and also less than the probability that the course exists. So models in **FuzzySet** allow the introduction of uncertainty in conceptual data models in a natural way.

In general, there are many issues related to uncertainty in data models. Information may not only be uncertain, but also missing, or vaguely known (*i.e.* only known to be in a certain set or interval of values). Missing information may have numerous interpretations, e.g. *not applicable*, *unknown*, *nonexistent*, or *confidential*. In [23] many of the issues related to imprecise and uncertain information are treated. The use of our approach could lead to the development of appropriate instance categories as a formal foundation for further study of these issues. Such instance categories at least have to incorporate features of the categories **FuzzySet** and **PartSet**.

In so-called historical or temporal databases the history of the creation and deletion of object instances is maintained (*see* e.g. [26], [29]). Historical databases do not forget information and allow previous states of the database to be restored, for example if performed changes turn out to be incorrect or undesirable. Conceptual data modeling techniques exist that support an explicit notion of time and consequently facilitate the implementation of an application in terms of a historical database management system. Examples of such techniques are described in [4] and [31]. The framework can provide a natural semantics for these techniques by the choice of an appropriate instance category. Such an instance category could be the category **TimeSet**, where each object is a set with a function assigning a time interval to each element of that set. The set of time points $\mathcal{T}$ is a totally ordered set and has a maximal element *now*. A time interval is an element $\langle t_1, t_2 \rangle$ from $\mathcal{T} \times \mathcal{T}$ such that $t_1 \leq t_2$. The morphisms in this category are functions between the sets of the associated objects such that for each element in the range of that function the time interval is included in the time interval of the origin of that element. This is a necessary requirement and, among others, reflects that composed objects can not outlive their various parts. For example, the fact that a certain person P_1 is enrolled in a certain course C_1 cannot be known before either P_1 or C_1 have come into existence, or after either P_1 or C_1 are no longer "alive".

7. CONCLUSIONS AND FURTHER RESEARCH

The approach that is described in this paper is a very general one that allows us to define the semantics of conceptual data models in a wide range of instance categories. The range of possible instance categories is much wider than those used in other semantics since it is also possible to use categories that are not topoi or cartesian closed. Further, since only simple category theory constructions have been used, our semantics is easily accessible.

This wide range of instance categories allows experimenting with extensions such as the handling of missing values (e.g. by using **PartSet**) or uncertainties (e.g. by using **FuzzySet**).

The range of valid type models is broader than in most other conceptual data models. There are very few restrictions on the type graphs. Therefore, it is possible to let fact types be subtypes of other types, or to make fact types act as power types. In fact all object types are treated very uniformly. This uniformity also helps in reducing the number of different concepts. For example, as was described in section 5.1 the collection cover constraint and the subtype cover constraint can be treated as special cases of the total role constraint. This is not possible with other conceptual data models.

The model that is described here is very similar to object-oriented data models. The subtypes in our approach are analogous to subclasses. Attributes can be modeled using roles. Attribute inheritance could be incorporated explicitly in the type model by adding an attribute that is defined in a given type to all its subtypes. The value of this subtype attribute arrow in the type model is the composition of the original attribute arrow with the subtype arrow from the subtype to the supertype. The resulting model is similar to that of [32].

As a last advantage we should mention that the use of category theory has been very helpful in discovering the essence of concepts in conceptual data modeling.

7.1. Further research

In [32], [8] the standard relational database operations are defined using category theory constructs. These authors used topoi as instance categories. Even though the current framework uses simpler categories, it still appears possible to define most standard relational database operations.

Although the uniqueness and total role constraints are by far the most important constraints used in conceptual data models, there are several others that would also be worth including. Examples are the exclusion, equality, and subset constraint.

PSM also contains sequence types that are similar to arrays (or lists) of objects. Although it has been omitted from this paper, it is possible to add these in a simple way to the type model. A more intriguing question is if it is possible to add arbitrary algebraic types, perhaps in a similar way to the "sketches" given in [10].

The current framework could also be extended to describe database updates and schema evolution.

ACKNOWLEDGMENTS

We like to thank Miranda Aldham-Breary for improving the English of this paper. Comments by Paul Frederiks, Denis Verhoef, Hans Zantema, and Henk Barendregt are appreciated.

A. APPENDIX: SAMPLE INSTANCE CATEGORIES

This appendix contains some proofs about the properties of several instance categories that have been used in the rest of the paper. In many publications **Set** or **FinSet** are used as prototypical examples of concrete categories; it is therefore somewhat surprising that it is difficult to find literature about related categories such as **Rel**, **PartSet** and **FuzzySet**.

A.1. The category Rel

The objects in the category **Rel** are sets, and the arrows $R : A \rightarrow B$ are binary relations such that $\mathbf{dom}R \subseteq A$ and $\mathbf{ran}R \subseteq B$.

Note that this category is different from a similar named category used by some other authors (e.g. [3]) in which the objects themselves are relations.

DEFINITION A.1:

1. $\mathbf{dom}R = \{a | \langle a, b\rangle \in R\}$
2. $\mathbf{ran}R = \{b | \langle a, b\rangle \in R\}$
3. $\mathbf{ID}_A = \{\langle a, a\rangle | a \in A\}$
4. $\tilde{R} = \{\langle b, a\rangle | \langle a, b\rangle \in R\}$
5. *The* relational image *of a set S under R is defined by*

$$R(|S|) = \{b \in \mathbf{ran}R | \exists_{s \in S}[\langle s, b\rangle \in R]\}.$$

□

PROPOSITION A.1: *The following properties are true for relations P, Q, R:*

1. *If R is a bijective relation* $R \circ \tilde{R} = \mathbf{ID}_{\mathbf{ran}R}$
2. $\mathbf{dom}R \subseteq A \Rightarrow R = R \circ \mathbf{ID}_A$
3. $\mathbf{ran}R \subseteq A \Rightarrow R = \mathbf{ID}_A \circ R$
4. $R \circ (P \cup Q) = (R \circ P) \cup (R \circ Q)$

A.1.1. *Monomorphisms and epimorphisms*

DEFINITION A.2: *A relation $R : A \rightarrow B$ is injective iff $R' \subseteq \tilde{R}$ exists such that $R' \circ R = \mathbf{ID}_A$.* □

LEMMA A.1: *A relation $R : A \rightarrow B$ in* **Rel** *is a monomorphism iff it is an injective relation.*

Proof: Suppose R is an injective relation. Let $P, Q : C \rightarrow A$ be relations such that $R \circ P = R \circ Q$, then

$$R \circ P = R \circ Q \Rightarrow R' \circ R \circ P = R' \circ R \circ Q \Rightarrow \mathbf{ID}_A \circ P = \mathbf{ID}_A \circ Q \Rightarrow P = Q$$

Hence R is a monomorphism.

Suppose that R is a monomorphism. For $a \in A$ let $P_a = \{c\} \times (A - \{a\})$. Let $Q = \{c\} \times A$. Since $P_a \neq Q$, and R is a monomorphism $R \circ P_a \neq R \circ Q$. As $R \circ P_a \subseteq R \circ Q$ an element b_a must exist such that

$$\langle c, b_a \rangle \in R \circ Q \wedge \langle c, b_a \rangle \notin R \circ P_a.$$

Obviously, $\langle a, b_a \rangle \in R$. Furthermore, $a' \in A - \{a\}$ can not exist such that $\langle a', b_a \rangle \in R$ because then $\langle c, b_a \rangle \in R \circ Q$. Let $R' = \{\langle b_a, a \rangle | a \in A\}$. Then $R' \subseteq \tilde{R}$ and $R' \circ R = \mathbf{ID}_A$. Therefore, R is an injective relation. □

DEFINITION A.3: *A relation $R : A \rightarrow B$ is surjective iff $R' \subseteq \tilde{R}$ exists such that $R \circ R' = \mathbf{ID}_B$.* □

LEMMA A.2: *A relation $R : A \rightarrow B$ in* **Rel** *is an epimorphism iff it is a surjective relation.*

Proof: The category **Rel** is its own dual category. Every arrow is mapped onto its converse by the dual functor. The dual of a monomorphism is an epimorphism and vice versa. Likewise, the dual of an injective relation is a surjective relation. Therefore, this lemma follows from lemma A.1 by duality. □

A.1.2. *Limits and colimits*

LEMMA A.3: *The dual of a colimit in a category C of a diagram D is the limit in the dual category of C of the dual of D.*

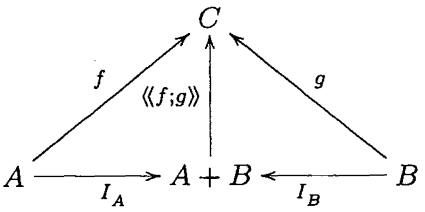
Proof: This follows immediately from the definitions of (co)limits and dual. □

LEMMA A.4: *The coproduct $A + B$ in the category* **Rel** *is equal to the disjoint sum of A and B.*

Proof: Let $f : A \to C$ and $g : B \to C$. Then for all f and g the following arrow can be constructed:

$$\langle\langle f; g\rangle\rangle : A + B \to C = \{\langle I_A(a), c\rangle | \langle a, c\rangle \in f\} \cup \{\langle I_B(b), c\rangle | \langle b, c\rangle \in g\}$$

It is immediate from the construction that the following diagram commutes:



That this arrow is unique can be seen as follows. Suppose that there is an arrow $h : A + B \to C$ such that $h \circ I_A = f$ and $h \circ I_B = g$.

Let $A' = \mathbf{ran} I_A$ and let $B' = \mathbf{ran} I_B$. Then

$$\begin{aligned} & h \circ I_A = f = \langle\langle f; g\rangle\rangle \circ I_A \\ \Rightarrow & h \circ I_A \circ \tilde{I}_A = \langle\langle f; g\rangle\rangle \circ I_A \circ \tilde{I}_A \\ \Rightarrow & h \circ \mathbf{ID}_{A'} = \langle\langle f; g\rangle\rangle \circ \mathbf{ID}_{A'} \end{aligned}$$

By using similar reasoning it can be shown that

$$h \circ \mathbf{ID}_{B'} = \langle\langle f; g\rangle\rangle \circ \mathbf{ID}_{B'}$$

Using the observation that $\mathbf{ID}_{A+B} = \mathbf{ID}_{A'} \cup \mathbf{ID}_{B'}$:

$$\begin{aligned} h &= h \circ \mathbf{ID}_{A+B} \\ &= h \circ (\mathbf{ID}_{A'} \cup \mathbf{ID}_{B'}) \\ &= h \circ \mathbf{ID}_{A'} \cup h \circ \mathbf{ID}_{B'} \\ &= \langle\langle f; g\rangle\rangle \circ \mathbf{ID}_{A'} \cup \langle\langle f; g\rangle\rangle \mathbf{ID}_{B'} \\ &= \langle\langle f; g\rangle\rangle \circ (\mathbf{ID}_{A'} \cup \mathbf{ID}_{B'}) \\ &= \langle\langle f; g\rangle\rangle \circ \mathbf{ID}_{A+B} \\ &= \langle\langle f; g\rangle\rangle \end{aligned}$$

Therefore, it can be concluded that $\langle\langle f; g\rangle\rangle$ is unique. □

LEMMA A.5: *Complementable monomorphisms in* **Rel** *are injective relations that are functions.*

Proof: Complementable monomorphisms are isomorphic to sum injection arrows that are functions. Isomorphisms in **Rel** must be both injective and surjective relations, in other words they must be bijective total functions. Since the composition of functional relations gives another functional relation complementable monomorphisms must be functions. □

The complement of a relation

$$f : A \rightarrow B \text{ is } \hat{f} : B - A \rightarrow B = \{\langle b, b\rangle | b \in B - f(|A|)\}.$$

LEMMA A.6: *The product* $A \times B$ *in* **Rel** *is equal to the (disjoint) sum and the product selection arrows are partial functions that are the inverses of the sum injection arrows.*

Proof: The product is the limit over a discrete diagram. The dual of such a diagram is the diagram itself. From lemmas A.3 and A.4 it follows that the object $A \times B$ must be equal to $A + B$, and that the product projection arrows must be the converse of the sum injection arrows. □

LEMMA A.7: *The category* **Rel** *does not have all limits.*

Proof: Let

$$f : A \rightarrow B = \{\langle a_1, b_1\rangle\}$$

and let

$$g : A \rightarrow B = \{\langle a_1, b_1\rangle, \langle a_2, b_1\rangle\}.$$

Suppose an equalizer $e : E \rightarrow A$ of f and g exists. The following arrow equalizes f and g, $h : H \rightarrow A = \{\langle h_1, a_1\rangle, \langle h_2, a_1\rangle, \langle h_2, a_2\rangle\}$. If e is an equalizer a unique arrow $q : H \rightarrow E$ must exist such that $e \circ q = h$. As $\langle h_1, a_1\rangle \in h$, $x \in E$ exists such that $\langle h_1, x\rangle \in q \wedge \langle x, a_2\rangle \in e$. Since $\langle h_1, a_2\rangle \notin h$ it follows that $\langle x, a_2\rangle \notin e$. Since $\langle h_2, a_2\rangle \in h$, $y \in E$ exists such that $\langle h_2, y\rangle \in q \wedge \langle y, a_2\rangle \in e$. Since $\langle x, a_2\rangle \notin e$ and $\langle y, a_2\rangle \in e$, x and y must be different. As $\langle y, b_1\rangle \in g \circ e$, $\langle y, b_1\rangle \in f \circ e$, but this is only possible if $\langle y, a_1\rangle \in e$. It is possible to construct a new arrow q' by adding or deleting the tuple $\langle h_2, x\rangle$ from q. In this case $e \circ q' = e \circ q = h$ but $q' \neq q$, thus the arrow q is not unique and e cannot be an equalizer of f and g. □

Since **Rel** does not have all limits it is not a topos.

LEMMA A.8: *The category* **Rel** *does not have all colimits.*

Proof: Follows immediately from lemmas A.3 and A.7. □

LEMMA A.9: *The empty set is both an initial and final object in* **Rel**.

Proof: The only possible arrow to or from the empty set is an empty relation. There is exactly one such relation with every other object in **Rel**. □

LEMMA A.10: **Rel** *is a valid instance category.*

Proof: A functor F : **FinSet** $\rightarrow$ **Rel** must be constructed such that for all M_1, $M_2 : \mathcal{G} \rightarrow$ **FinSet** it holds that $M_1 = M_2 \Leftrightarrow F \circ M_1 = F \circ M_2$. As **FinSet** is a subcategory of **Rel** the inclusion functor obviously satisfies this requirement. □

A.2. The category PartSet

The category **PartSet** has sets as objects and partial functions as arrows.

The notation $f(x) \downarrow$ means that f is defined at x and similarly $f(x) \uparrow$ means that f is not defined at x.

A.2.1. *Monomorphisms and epimorphisms*

LEMMA A.11: *An arrow* $R : A \rightarrow B$ *in* **PartSet** *is a monomorphism iff it is an injective total function.*

Proof: The proof is similar to that of lemma A.1. □

LEMMA A.12: *An arrow* $R : A \rightarrow B$ *in* **PartSet** *is an epimorphism iff it is a surjective total function.*

Proof: The proof is analogous to the proof of proposition 2.9.2 in [10]. □

A.2.2. *Limits and colimits*

LEMMA A.13: *The coproduct* $A + B$ *in the category* **PartSet** *is equal to the disjoint sum (in* **Set***) of* A *and* B.

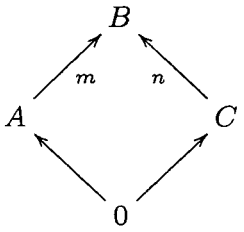
Proof: The proof is similar to that of lemma A.4. □

LEMMA A.14: *Coproducts in* **PartSet** *are disjoint.*

Proof: This is obvious because the coproduct is equal to the disjoint sum. □

LEMMA A.15: *All monomorphisms in* **PartSet** *are complementable.*

Proof: Let $m : A \rightarrow B$ be a monomorphism. Let $C = B - m(|A|)$, and let $n : C \rightarrow B = \mathbf{ID}_C$. Then the following diagram is both a pushout and a pullback:



□

LEMMA A.16: *The product* $A \times B$ *in* **PartSet** *is equal to:*

$$A \breve{\times} B \cup A \breve{\times} \{\perp\} \cup \{\perp\} \breve{\times} B$$

where $\perp$ *is an element that does not occur in* A *or* B, *and* $\breve{\times}$ *is the standard cartesian product in* **Set**. *The product selection arrows are defined by:* $\pi_A : A \times B \rightarrow A = \{\langle a, b\rangle \mapsto a | \langle a, b\rangle \in A \times B \wedge a \neq \perp\}$ *and* $\pi_B : A \times B \rightarrow B = \{\langle a, b\rangle \mapsto b | \langle a, b\rangle \in A \times B \wedge b \neq \perp\}$.

Proof: It is not difficult to show that π_A and π_B have the following properties:

$$\forall_{a \in A, b \in B} \exists!_{t \in A \times B} [\pi_A(t) = a \wedge \pi_B(t) = b]$$

$$\forall_{a \in A} \exists!_{t \in A \times B} [\pi_A(t) = a \wedge \pi_B(t) \uparrow]$$

$$\forall_{b \in B} \exists!_{t \in A \times B} [\pi_A(t) \uparrow \wedge \pi_B(t) = b]$$

$$\neg \exists_{t \in A \times B} [\pi_A(t) \uparrow \wedge \pi_B(t) \uparrow]$$

Further, for any partial functions f, g, h such that

$$h = f \circ g : h(x) \downarrow \Leftrightarrow g(x) \downarrow \wedge f(g(x)) \downarrow .$$

Let $f : C \rightarrow A$ and $g : C \rightarrow B$ be two arrows. If $A \times B$ is indeed a product a unique arrow u must exist such that $\pi_A \circ u = f$ and $\pi_B \circ u = g$. For each $c \in C$ these requirements determine a unique value for $u(c)$ as follows:

• $f(c) \downarrow \wedge g(c) \downarrow$: Then $u(c) \downarrow$, since $\pi_A(u(c)) \downarrow$, and $u(c) = \langle f(c), g(c)\rangle$ is the unique value for which $\pi_A(u(c)) = f(c)$ and $\pi_B(u(c)) = g(c)$.

• $f(c) \downarrow \wedge g(c) \uparrow$: Then $u(c) \downarrow$, since $\pi_A(u(c)) \downarrow$, and $u(c) = \langle f(c), \perp \rangle$ is the unique value for which $\pi_A(u(c)) = f(c)$ and $\pi_B(u(c)) \uparrow$.

• $f(c) \uparrow \wedge g(c) \downarrow$: Then $u(c) \downarrow$, since $\pi_B(u(c)) \downarrow$, and $u(c) = \langle \perp, g(c) \rangle$ is the unique value for which $\pi_A(u(c)) \uparrow$ and $\pi_B(u(c)) = g(c)$.

• $f(c) \uparrow \wedge g(c) \uparrow$: As $\neg \exists_{t \in A \times B} [\pi_A(t) \uparrow \wedge \pi_B(t) \uparrow]$ it must be the case that $u(c) \uparrow$. □

LEMMA A.17: *Let $f, g : A \rightarrow B$ be two arrows in* **PartSet**. *Let $R = \{\langle f(a), g(a) \rangle | a \in \mathbf{dom} f \cap \mathbf{dom} g\}$, and let $\tilde{R}$ be the reflexive, symmetric, transitive closure of R. Let U be the set of equivalence classes modulo $\tilde{R}$ and let $h : B \rightarrow U$ be the function that maps an element from B to its equivalence class. Then h is a coequalizer of f and g.*

Proof: See the proof of proposition 8.4.2 in [10]. □

LEMMA A.18: *The equalizer in* **PartSet** *of two arrows $f, g : A \rightarrow B$ is $E = \{a \in A | f(|\{a\}|) = g(|\{a\}|)\}$ and $e : E \rightarrow A = \mathbf{ID}_E$.*

Proof: It is easy to check that $f \circ e = g \circ e$.

Let $h : C \rightarrow A$ be such that $f \circ h = g \circ h$. Since h equalizes f and g we have $\mathbf{ran} h \subseteq E$. Hence, the arrow $k : C \rightarrow E$ can be constructed as $k = h$. Since $\mathbf{ran} k \subseteq E$, $e \circ k = \mathbf{ID}_E \circ k = k = h$.

Assume that there is an arrow $k' : C \rightarrow E$ such that $e \circ k' = h$. Since e is a total injective function it is a monomorphism. From $h = e \circ k' = e \circ k$ it can be concluded that $k' = k$, in other words k is unique. □

LEMMA A.19: *The empty set is both an initial and final object in* **PartSet**.

Proof: The only possible arrow to or from the empty set is an empty function. There is exactly one such function with every other object in **PartSet**. □

LEMMA A.20: *The category* **PartSet** *has all finite limits and colimits.*

Proof: From the lemmas A.19, A.16, and A.18 we know that **PartSet** has a final object, all products and equalizers, therefore it has all finite limits. From the lemmas A.19, A.13, and A.17 we know that **PartSet** has an initial object, all sums and coequalizers, therefore it has all finite colimits. □

This also follows from the equivalence between **PartSet** and the category **pSet** of pointed sets. For the equivalence refer to [10] and for the completeness refer to [3].

LEMMA A.21: **PartSet** *is a valid instance category.*

Proof: For the proof a functor F : **FinSet** $\rightarrow$ **PartSet** must be constructed such that for all M_1, $M_2 : \mathcal{G} \rightarrow$ **FinSet** it holds that

$$M_1 = M_2 \Leftrightarrow F \circ M_1 = F \circ M_2.$$

Since **FinSet** is a subcategory of **PartSet** the inclusion functor obviously satisfies this requirement. □

A.3. The category FuzzySet

Let P be a complete Heyting algebra. The notations 0 and 1 will be used to indicate the minimal and maximal element in P. The objects in the category **FuzzySet** are sets S together with a function $\sigma_S : S \rightarrow P$. The morphisms $f : (S, \sigma_S) \rightarrow (T, \sigma_T)$ are functions $f : S \rightarrow T$ such that $\sigma_S \leq \sigma_T \circ f$.

A.3.1. *Monomorphisms and epimorphisms*

LEMMA A.22: *Epimorphisms in* **FuzzySet** *are surjective functions, while monomorphisms in* **FuzzySet** *are injective functions.*

Proof: Similar to the proofs in **Set**. □

A.3.2. *Limits and colimits*

LEMMA A.23: *The coproduct of two objects A and B is (C, σ_C) where C is the disjoint union of A and B and $\sigma_C(x) = \sigma_A(x)$ for elements x that have been injected from A and $\sigma_C(x) = \sigma_B(x)$ for elements x that have been injected from B.*

LEMMA A.24: *The product of two objects A and B is (C, σ_C) where C is the cartesian product of A and B and $\sigma_C(\langle a, b \rangle) = \sigma_A(a) \sqcup \sigma_B(b)$.*

LEMMA A.25: *The coequalizer of two arrows $f, g : (A, \sigma_A) \rightarrow (B, \sigma_B)$ is $h : (B, \sigma_B) \rightarrow (E, \sigma_E)$ where E together with h is the (**Set**) coequalizer of f and g. Let $I_e = \{b \in B | h(b) = e\}$ then σ_E is defined by $\sigma_E(e) = \sqcap_{b \in I_e} \sigma_B(b)$.*

LEMMA A.26: *The equalizer of two arrows $f, g : (A, \sigma_A) \rightarrow (B, \sigma_B)$ is $h : (E, \sigma_E) \rightarrow (A, \sigma_A)$ where E together with h is the (**Set**) equalizer of f and g. σ_E is defined by $\sigma_E(e) = \sigma_A(h(e))$.*

LEMMA A.27: *The initial object in* **FuzzySet** *is $(\emptyset, \sigma_\emptyset)$ where $\sigma_\emptyset$ is the empty function.*

LEMMA A.28: *The final object in* **FuzzySet** *is* $(\{*\}, \sigma_{\{*\}})$ *where* $\{*\}$ *is a one element set, and* $\sigma_{\{*\}}(*) = 1$.

LEMMA A.29: **FuzzySet** *is a valid instance category.*

Proof: For the proof a functor F : **FinSet** $\rightarrow$ **FuzzySet** must be constructed such that for all M_1, M_2 : $\mathcal{G} \rightarrow$ **FinSet** it holds that $M_1 = M_2 \Leftrightarrow F \circ M_1 = F \circ M_2$. If we define $F(S) = (S, \lambda s \in S.1)$ this requirement is obviously satisfied. □

REFERENCES

1. D. E. AVISON and G. FITZGERALD, *Information Systems Development: Methodologies, Techniques and Tools*, Blackwell Scientific Publications, Oxford, United Kingdom, 1988.
2. S. ABITEBOUL and R. HULL, IFO: A Formal Semantic Database Model, *ACM Transactions on Database Systems*, 1987, *12*(4), pp. 525-565.
3. J. ADÁMEK, H. HERRLICH and G. E. STRECKER, *Abstract and Concrete Categories*, Pure and applied mathematics, John Wiley & Sons, New York, 1990.
4. G. ARIAV, A Temporally Oriented Data Model, *ACM Transactions on Database Systems*, 1986, *11*(4), pp. 499-527.
5. C. W. BACHMAN, Data structure diagrams, *Data Base*, 1969, *1*(2), pp. 4-10.
6. M. BARR, Fuzzy sets and topos theory, *Canadian Mathematical Bulletin*, 1986, *24*, pp. 501-508.
7. M. L. BRODIE, On the development of data models. In M. L. Brodie, J. Mylopoulos and J. W. Schmidt, editors, *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases and Programming Languages*, pp. 19-48. Springer-Verlag, New York, 1984.
8. K. BACLAWSKI, D. SIMOVICI and W. WHITE, A categorical approach to database semantics, *Mathematical Structures in Computer Science*, 1994, *4*, pp. 147-183.
9. J. A. BUBENKO, Information System Methodologies - A Research View. In T. W. Olle, H. G. Sol and A. A. Verrijn-Stuart, editors, *Information Systems Design Methodologies: Improving the Practice*, pp. 289-318. North-Holland, Amsterdam, The Netherlands, 1986.
10. M. BARR and C. WELLS, *Category Theory for Computing Science*, Prentice-Hall, Englewood Cliffs, New Jersey, 1990.
11. P. P. CHEN, The Entity-Relationship Model: Toward a Unified View of Data, *ACM Transactions on Database Systems*, 1976, *1*(1), pp. 9-36.
12. E. F. CODD, A Relational Model of Data for Large Shared Data Banks, *Communications of the ACM*, 1970, *13*(6), pp. 377-387.
13. G. ENGELS, M. GOGOLLA, U. HOHENSTEIN, K. HÜLSMANN, P. LÖHR-RICHTER, G. SAAKE and H.-D. EHRICH, Conceptual modelling of database applications using an extended ER model, *Data & Knowledge Engineering*, 1992, *9*(4), pp. 157-204.
14. J. J. VAN GRIETHUYSEN, editor, *Concepts and Terminology for the Conceptual Schema and the Information Base*, Publ. nr. ISO/TC97/SC5-N695, 1982.
15. T. A. HALPIN, *Conceptual Schema and Relational Database Design*, Prentice-Hall, Sydney, Australia, 2nd edition, 1995.

16. M. HAMMER and D. MCLEOD, Database Description with SDM: A Semantic Database Model, *ACM Transactions on Database Systems*, 1981, *6*(3), pp. 351-386.
17. T. A. HALPIN and M. E. ORLOWSKA, Fact-oriented modelling for data analysis, *Journal of Information Systems*, 1992, *2*(2), pp. 97-119.
18. A. H. M. TER HOFSTEDE, H. A. PROPER and Th. P. VAN DER WEIDE, Formal definition of a conceptual language for the description and manipulation of information models, *Information Systems*, 1993, *18*(7), pp. 489-523.
19. A. H. M. TER HOFSTEDE and Th. P. VAN DER WEIDE, Expressiveness in conceptual data modelling, *Data & Knowledge Engineering*, 1993, *10*(1), pp. 65-100.
20. C. B. JONES, *Systematic Software Development using VDM*, Prentice-Hall, Englewood Cliffs, New Jersey, 1986.
21. W. KIM and F. H. LOCHOVSKY, editors, *Object-Oriented Concepts, Databases, and Applications*, ACM Press, Frontier Series. Addison-Wesley, Reading, Massachusetts, 1989.
22. D. MAIER, *The Theory of Relational Databases*, Computer Science Press, Rockville, Maryland, 1988.
23. J. M. MORRISSEY, Imprecise Information and Uncertainty in Information Systems, *ACM Transactions on Information Systems*, 1990, *8*(2), pp. 159-180.
24. G. M. NIJSSEN and T. A. HALPIN, *Conceptual Schema and Relational Database Design: a fact oriented approach*, Prentice-Hall, Sydney, Australia, 1989.
25. A. PITTS, Fuzzy sets do not form a topos, *Fuzzy Sets and Systems*, 1982, *8*, pp. 101-104.
26. R. SNODGRASS and I. AHN, Temporal Databases, *IEEE Computer*, 1986, *19*(9), pp. 35-42.
27. D. W. SHIPMAN, The Functional Data Model and the Data Language DAPLEX, *ACM Transactions on Database Systems*, 1981, *6*(1), pp. 140-173.
28. A. SIEBES, *On Complex Objects*, PhD thesis, University of Twente, Enschede, The Netherlands, 1990.
29. R. SNODGRASS, Temporal Databases Status and Research Directions, *SIGMOD Record*, 1990, *19*(4), pp. 83-89.
30. J. M. SPIVEY, *Understanding Z: A Specification Language and its Formal Semantics*, Cambridge University Press, Cambridge, United Kingdom, 1988.
31. C. THEODOULIDIS, P. LOUCOPOULOS and B. WANGLER, A conceptual modelling formalism for temporal database applications, *Information Systems*, 1991, *16*(4), pp. 401-416.
32. C. TUIJN, *Data Modeling from a Categorical Perspective*, PhD thesis, University of Antwerp, Antwerp, Belgium, 1994.
33. T. J. TEOREY, D. YANG and J. P. FRY, A logical design methodology for relational databases using the extended entity-relationship model, *ACM Computing Surveys*, 1986, *18*(2), pp. 197-222.
34. S. B. ZDONIK and D. MAIER, editors, *Readings in Object-Oriented Database Systems*, Morgan Kaufmann, San Mateo, California, 1990.