

# FRAME INFORM SYSTEM

LDBD Research Report ◇ May 1995

## Algebraic Graph-Based Approach to Management of Multi-Base Systems,II: Mathematical Aspects of Schema Integration

**Zinovy Diskin**

Laboratory for Database Design  
FIS/LDBD-95-02, May 1995

# Algebraic graph-based approach to management of multi-base systems, II: Mathematical aspects of schema integration

Zinovy Diskin  
Frame Inform Systems, Riga  
diskin@frame.riga.lv

## 1 Introduction

Conceptual design and schema integration are among central issues in the DB area. Their significance is commonly recognized and to date has increased greatly due to the evident tendency of organizing modern (and of the nearest future) information systems on cooperative principals. Indeed, for cooperative systems (CIS) it is normal that an application program needs data stored in several separate databases (DBs). In addition, a peculiarity of CIS consists in heterogeneity of local DB systems caused by orientation at different level applications, and/or by the autonomy of their origin and initial development. So, heterogeneous multibase integration appears as a fundamental issue of CIS functioning. In particular, in the context of federated database systems (FDBS), integration is a function regularly performed at different levels and by different services depending on the organization of the FDBS environment.

The value of the problem is well known, various approaches, techniques and sometimes tools were proposed (see, eg, [7, 22, 35, 36, 30, 29, 26] and surveys [2, 27]). In spite of the diversity of approaches, several common points can be well identified. Integration consists of *schema integration* – composing a global schema from the set of local ones, and *data integration* – computing virtual extension of the global schema. Schema integration is the key issue: its correctness determines the correctness and effectiveness of the second phase.

Despite the large body of work done in the area, several fundamental questions remain unanswered, moreover, often they have not been stated at all. In particular, it remains open whether a given view definition language or a correspondence assertions language are sufficient for specifying the variety of all practically possible situations of local DB overlapping.

The question is immediately connected with an extremely important issue of classifying conflicts between local schemas in their representation of the overlapping information. It is well known that the same data can be structured in different ways, and resolving such structural conflicts between representations is the key for correct integration. In fact, a methodology of integration is mainly determined by the

treatment and the taxonomy of conflicts it adopts. Several approaches were proposed ([7, 19, 28, 30] and others), the most fundamental to date research is [29] where a taxonomy distinguishing *semantic*, *descriptive*, *structural* and *heterogeneity* conflicts was developed. It appears however that this is rather a substantial description of various situations leading to conflicts (and, indeed, it properly encompasses the diversity) than a precise formal specification capable to support automated integration.

Speaking in general, it appears that resolution of structural conflicts within a sufficiently rich data model is still a challenge. The more so is for heterogeneous conflicts where it seems little was done apart from setting the problem and several declarations.

In the paper [6] we have proposed a new approach to schema and database integration based on a special graphical yet formalized specification language generalizing the so called *sketches* developed in categorial logic ([1] is a standard reference). In this approach a conceptual database schema is a *generalized sketch*, that is, a directed multigraph (further simply a graph) in which some diagrams (of nodes and arrows) are labeled by special markers. In addition, queries against the schema are coded by *diagram operations* which are denoted by labeling corresponding diagrams with special *operation markers*. In other words, we have a graphical mechanism for denoting query-determined derived items of a conceptual schema.

On this basis we have shown that structural and heterogeneity conflicts between local schemas are reduced to well understood difference between basic and derived data: items of the conceptual schema of one view considered there as basic can be safely considered as derived in another view. On the other hand, semantic and descriptive distinctions between views can be specified by an additional *correspondence information sketch*,  $S_{CI}$ . We have demonstrated that in such a setting overlapping between views can be completely specified by a finite set of equations,  $E_{CI}$ , between items of local sketches (including  $S_{CI}$ ). The sketch  $S_{CI}$  is syntactically similar to local schemas so that integration of  $n$  local schemas turns into disjoint merging  $n + 1$  sketches,  $S_1, \dots, S_n, S_{n+1}$  with  $S_{n+1} = S_{CI}$ , and factorizing the result by the congruence generated by  $E_{CI}$  (ie, gluing together certain items of the merge according to the  $E_{CI}$ -equations).

Quasi-formally, the result of the above-mentioned manipulations can be described as

$$\overline{S_I} = (\overline{S_1} \oplus \dots \oplus \overline{S_n} \oplus \overline{S_{CI}}) / E_{CI}$$

where  $\overline{S_i}$ 's denote results of extending local sketches with derived items and  $\overline{S_I}$  is the result of merging.

Actually this procedure is performed in two steps. The first one consists in disjoint merging ( $n+1$ ) graphs underlying local sketches and then factorizing the merge according to  $E_{CI}$ -equations (this step can be performed in a fully automatic way). The second step consists in converting the integrated graph into a sketch integrating diagram markers from the local sketches. However, here diagram conflicts can arise, and these are real conflicts between views which can bring to light inconsistency of views!

Together with the integrated sketch  $\overline{S_I}$ , the integration procedure determines mappings from local sketches

$$\overline{\sigma_i}: \overline{S_i} \rightarrow \overline{S_I}$$

. In addition, images of these mappings cover  $\overline{S_I}$  so that each of its nodes or arrows belongs to  $\overline{\sigma_i}(\overline{S_i})$  for at least one  $i$ .

The present paper is the companion of [6] aiming at a more mathematically developed presentation of schema integration. Indeed, it was demonstrated in [6] that the problem admits a natural algebraic treatment on both levels: the external level of metadata modeling (algebra of schemas and views) and the internal level of data modeling (algebra of schema and database components, *eg*, relational algebra).

The relevance of algebraic framework for the internal aspects of DB theory is well known and goes back to the classical Codd's papers. In particular, a consistent algebraic basis for the high-level theoretical foundation of relational DB theory was developed in [24]; and algebraic treatment of more application-oriented issues was displayed in [31]. However, semantic issues like integration need graph-oriented algebraic machinery. Elaboration of the corresponding framework of basic definitions is a focus of the present paper (see also [13]). We emphasize however that as it is demonstrated in [6] the machinery is extremely useful also for practical tasks, and makes it possible to extract a concrete methodology and algorithm for automated heterogeneous schema integration. We have called the approach *AGO* since its two principal characteristics are Algebraicity and Graph-Orientedness.

As for the external level of metadata modeling, only few attempts were made to develop a formalized metatheory of databases. Among them are papers [18, 33, 10, 15, 32] where a mapping-oriented (arrow) approach to the problem was suggested (see also a manifest [4]). In the present paper we show that the arrow machinery allows to formulate precisely multi-base interoperability issues in a polymorphic style when data models are parameters. For example, the expression for  $\overline{S_I}$  above can be considered as an abbreviation of a certain composition of operations in the metagraph of schemas which are commutative and associative. This immediately proves the latter properties for the integration operation as a whole (which might be compared with rather involved reasoning on this fact in [3]).

The rest of the paper is organized as follows. In section 2 we present several concrete examples of semantic modeling via sketches, in particular, familiar aggregation, generalization and grouping constructs. A simple demonstrating example of schema integration is considered in section 3. In section 4 a mathematical framework for the integration problem is discussed and some principal lines are pointed out. Methodologically, this section is the focus of the paper. The last three sections present basic definitions and statements in each of directions outlined in section 4. Among them the most mathematically interesting is section 5 where construction of composing diagram operations is examined

and a graph-based parsing is introduced. The latter provides theoretical basis for an intelligent graphical editor for schema integration.

**Acknowledgements.** The subject of the paper was numerously discussed during our seminars on DB design at Frame Inform Systems. I am indebted to Sergey Ageshin, Ilya Beylin and Boris Cadish for fruitful discussions and stimulating criticism.

Special thanks to Ol'ga Tkacheva for TeXing a lot of diagrams contained in the paper.

## 2 Data modeling via sketches

Even an overall glance at graphical images currently used in semantic modeling shows an abundance of various kinds of conventional graphical constructs, symbols, labels and markers. Every specialist in semantic modeling and every DB designer uses that kind of semantic schemas which one finds more suitable and convenient for one's purposes. This is natural as well as reasonable, however, some problems arise.

First of all, a natural theoretical question whether the existing collection of modeling constructs is sufficient for specifying all possible semantic constructions, *ie*, whether it is complete in some sense, and what 'sense' should we mean here? Another problem is the universality of view integration techniques and tools, that is, their independence of data model. At last, the field strongly needs a data definition/manipulation language combining evidence and user-friendliness of graph-oriented object semantic models together with formal rigor and clearness of relational models.

We firmly believe that the sketch-based approach provides a proper (with respect to managing heterogeneity) universal framework for data modeling and view integration. By suggesting this idea we do not want to force all DB designers to use the same universal specification language; let everyone use that collection of markers he likes. What we actually suggest concerns *what must be marked* in semantic schemas.

Indeed, in the majority of semantic schemas in use, particularly, in ER-diagrams which became a kind of the *de facto* standard in the area of conceptual design and modeling, in order to specify the intended semantic meaning of a given node  $N$  of a schema, one marks the very node  $N$  by a corresponding label. For instance, in the ER-standard to specify a node as a relation, *ie*, as a set of relationship objects (tuples), one labels it by a diamond. However, if another user perceives the same data objects as entities, he/she labels the corresponding node in his schema by a rectangle. How must be the node  $N$  of the integrated schema labeled?

One can observe that structural conflicts like above are caused by determining internal structure of a set via determining the structure of its elements: a relation is a set of tuples, a powerset is a set of subsets etc.

In contrast to thinking in terms of elements, the category theory paradigm of arrow thinking suggests to specify internal structure of elements of a given set by characterizing (labeling) the corresponding diagram of functions adjoint to the set. Here is a very simple example: sketch specification of relations.

Under a *source* we will mean a set, say,  $X$ , equipped with a family  $\mathcal{F}$  of functions,  $f_i: X \rightarrow Y_i$ ,  $i = 1, \dots, n$ . Then one has the function  $f = [f_1 \dots f_n]: X \rightarrow Y_1 \times \dots \times Y_n$  determined in the standard way:  $fx = [f_1x, \dots, f_nx]$ . It is easy to see that  $f$  will be injective, *ie*,  $X$  will be actually a

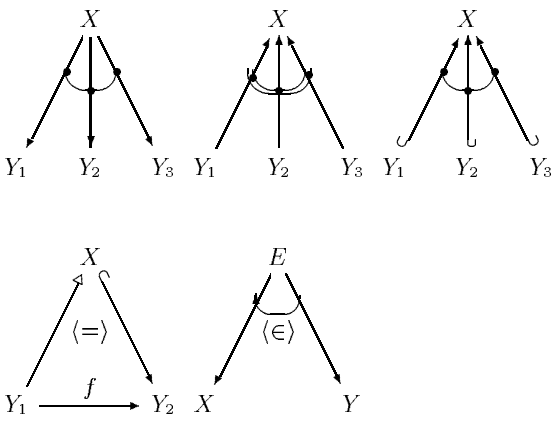


Figure 1: Several simple sketches

relation up to isomorphism, iff the family  $\mathcal{F}$  is *separating*, that is, satisfies the following *separation* condition:  $\forall x, x' \in X, x \neq x'$  implies  $f_i(x) \neq f_i(x')$  for some  $f_i$ .

So, to specify a set as a set of tuples one can safely leave the very set without imposition of any constraints but constrain instead the corresponding source of outgoing functions to be separating.

Correspondingly, on the syntax level, to specify a node as a relation one can safely leave the node without any marking but mark instead the corresponding source of outgoing arrows by a label (say, an arc) denoting the separation condition (actually, this is nothing but a well known idea of designating a key of a relation).

Thus, the leftmost sketch on Fig.1 describes the node  $X$  as a ternary relation. We mean that for any extent mapping  $\varepsilon$  which assigns sets  $X^\varepsilon, Y_j^\varepsilon$  to the corresponding nodes, and functions  $f_j^\varepsilon$  to the corresponding arrows ( $j = 1, 2, 3$ ), if  $\varepsilon$  satisfies the constraint expressed by the arc marker, *ie*, if  $(f_j^\varepsilon, j = 1, 2, 3)$  is a separating family, then  $X^\varepsilon$  is a relation (up to isomorphism). Further we will explain sketch markers in a semi-formal way without accurate distinguishing syntax and semantics levels. It is hoped that precise formulations can be easily built according to the example above.

A collection of diagram markers is presented in Table 1. Then, for example, the following five pictures (sketches) on Figure 1 explain the node  $X$  as a relation, disjoint sum (coproduct), union, the image of a given function, and a subset of the powerset (of  $Y$ ) respectively.

A great advantage of the sketch specification language is that it is (provably!) universal: it follows from general results of categorical logic that any formal data specification<sup>1</sup> can be replaced by an equivalent sketch of some appropriate type. So, sketches provide a real possibility to handle heterogeneity in a consistent (and effective, as we see below) way.

In addition, rigor semantics of sketches makes it possible to distinguish in the general integration procedure those parts which can be fully automated and those that have to be performed by an integrating person (DBA). In its turn, the latter phases can be also computer-aided and, on the whole, the sketch framework brings to light limitations and possibilities of automation in schema and data integration.

<sup>1</sup>That is, such a specification which, in principle, can be written down by a conventional formal language of the modern mathematics

On the other hand, our experience has shown that sketches turn out to be very handy as a machinery for semantic modeling and integration. In particular, they are convenient for semi-automated phases of integration being performed in an interactive mode. In addition, since the sketch treatment of integration is essentially algebraic several extensive integration steps can be reduced to formal algebraic manipulations with terms and equations which provides their effective computer realization.

### 3 An example of view integration via sketches and equations

To demonstrate main features of the approach we suggest we will consider a slight modification (convenient for expository purposes) of the "library example" of schema integration described in the survey [2].

Two views (universes of discourse) are presented by ER-diagrams depicted on Fig.2. We suppose further that views are overlapped as follows: 'BOOK's are 'PUBLICATION's and semantics of the left view relation 'BOOK' - 'PUBLISHER' coincides with semantics of the right view attribute 'Publisher'. Further, 'TOPIC's are embedded into 'KEYWORD's in a one-one way such that their attributes 'Name' and 'Code' coincide and instances of relationship between 'BOOK's and 'TOPIC's are also relationships between 'PUBLICATION' and 'KEYWORD'.

The key question for automated integration is how to express the information describing schema correspondence in a formal way so that it can be put into computer.

First of all, we convert ER-diagrams into sketches as presented on Fig.3 and add to them the sketch specifying that "books are publications". Nodes with typewriter names are predefined *value domains* whose semantics is *a priori* known to the DBMS. Rectangle nodes are *abstract classes* whose extensions should be stored in the DB.<sup>2</sup>

Now, to specify that semantics of the left view relation from 'BOOK's to 'PUBLISHER's coincides with semantics of the right view attribute 'Publisher' we proceed as follows.

We extend the correspondence sketch with derived arrow  $pbr^\#$  equal to the composition  $bap; publr$  (Fig.4): marker  $(=)$  hung on the corresponding diagram just denotes this definition. Then, in order to obtain a relation between 'Book's and 'Publication's we apply the operation  $PB$  to the couple of arrows  $(pbr^\#, pname)^\#$ <sup>3</sup> and then specify that the resulting relation  $(pbr'', bk'')$  is just the relation 'B-P' from the left schema by writing equations (4,5,6) into the table of *correspondence equations* on Fig.4

It would be convenient to picture initial sketches by one colour, say, black, and their derived components - by another colour, say, green. For typographical reasons, we replace green-painting derived items by hanging various superscript (like ' , " , \* , # etc.) on their names. In addition, derived nodes and arrows obtained simultaneously by applying some operation are marked with the same superscript, and the same superscript labels the very marker of the operation.

<sup>2</sup>For AGO, Int and {M,F} are *markers* (in our precise sense), hung on corresponding nodes, that is, constraints imposed on their intended semantic interpretations. At the same time, 'Book' is a *name* labeling the node without imposing any constraints.

<sup>3</sup>The sketch notation for some graph-based operations is presented in Tabl.???. Each operation is specified by its *output sketch*,  $S_{out}$ , containing a designated *input* subs sketch,  $S_{in}$ . Semantics is as follows: if an extension of  $S_{in}$  is given, then extensions of the output items belonging to  $S_{out} \setminus S_{in}$  can be computed by the corresponding procedure. Thus, Table ?? presents a graph-based query language.

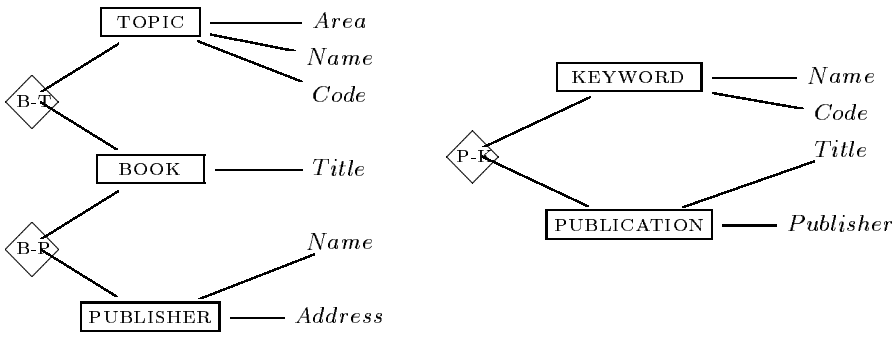


Figure 2: Two ER-schemas to be integrated

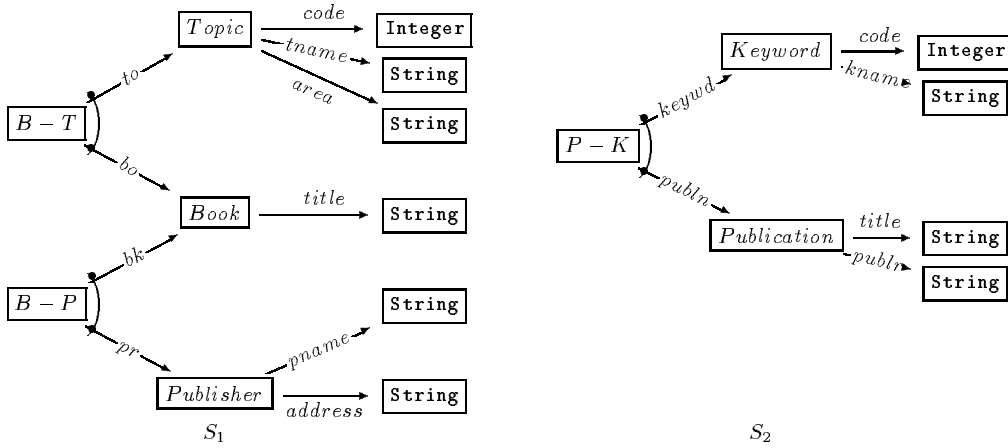
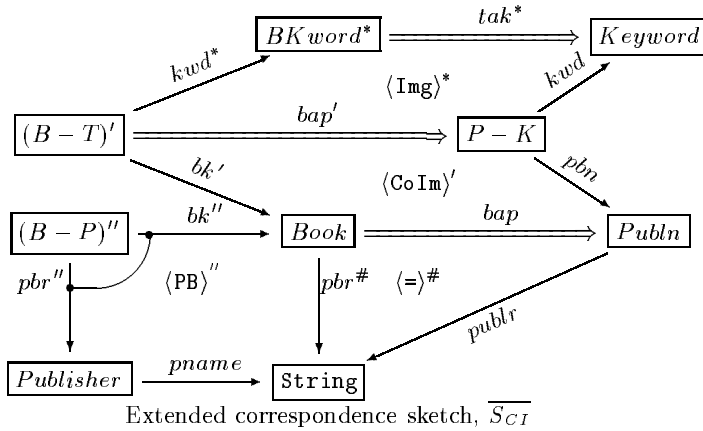


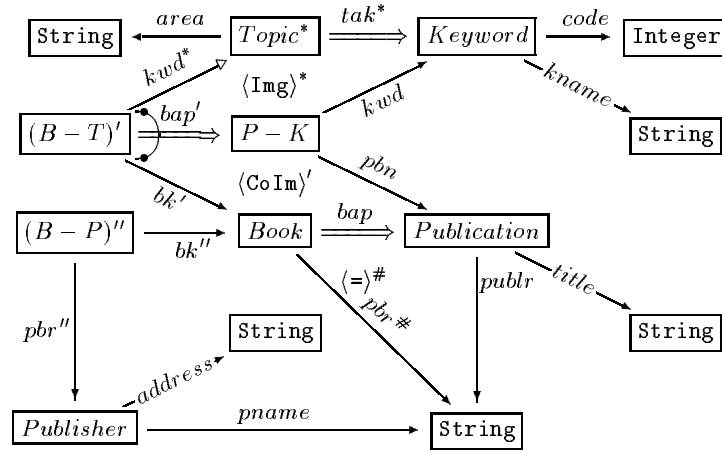
Figure 3: Transforming ER-schemas into Sketches



No	$\overline{S_{CI}}$	$S_1$	$S_2$
1	Book	Book	
2	Publn		Publication
3	publr		publr
4	$(B - P)''$	$(B - P)$	
5	$bk''$	bk	
6	$pbr''$	pr	
7	Keyword		Keyword
8	kwd		keywd
9	pbn		publn
10	$bk'$	bo	
11	$kwd^*$	to	
12	$bap; title$	title	
13	$tak^*; code$	code	
14	$tak^*; kname$	tname	

Correspondence equations,  $\overline{E_{CI}}$

Figure 4: Specifying Correspondence Information via Equations

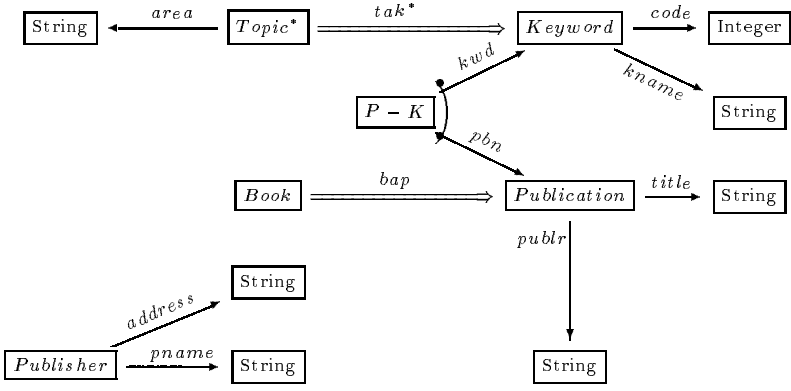


Mapping  $\sigma_1: S_1 \rightarrow \overline{S_I}$  Mapping  $\sigma_2: S_2 \rightarrow \overline{S_I}$

$S_1$	<i>to</i>	<i>bo</i>	<i>bk</i>	<i>pr</i>	<i>code</i>	<i>name</i>	<i>title</i>	<i>...</i>	$S_2$	<i>keywd</i>	<i>publn</i>	<i>publr</i>	<i>title</i>	<i>...</i>
$\overline{S_I}$	<i>kwd*</i>	<i>bk'</i>	<i>bk*</i>	<i>pbr*</i>	<i>tak*; code</i>	<i>tak*; kname</i>	<i>bap; title</i>	<i>...</i>	$\overline{S_I}$	<i>kwd</i>	<i>pbn</i>	<i>pbr</i>	<i>title</i>	<i>...</i>

Full integrated sketch,  $\overline{S_I}$ , and mappings of local sketches to it

Figure 5: Integration, A: Merging and Factorizing



Generating sketch,  $S_I$

$$Topic = Im(bap'; kwd) = Im(a.CoIm(pbn, bap); kwd)$$

Equational constraints

Figure 6: Integration, B: Removing Redundant Derived Items

In the similar way we specify the remaining part of the correspondence information by augmenting sketches with derived items and writing correspondence equations into the table as presented on Fig.4.

Thus, we have specified correspondence between views in a formal way and, moreover, our correspondence assertions are equations - this point has far going consequences.

We would like to emphasize that neither the correspondence sketch  $S_{CI}$  nor the set of correspondence equalities  $E_{CI}$  cannot be built automatically. Only the integrating person (DBA) possesses the information on inter-schema correspondence and carries the responsibility of specifying this information, that is, of setting the sketch  $S_{CI}$  and equalities  $E_{CI}$ . However, the very process of building sketches can be supported by an intelligent graphical editor. In particular, augmentation of sketches with derived items can be assisted by a built-in parsing checker for verifying correctness of hanging operation markers: input items of an operation should be among output items of preceding operations (we will return to this point at section??).

The next step of the integration process can be performed automatically. It consists in glueing together those nodes and arrows that appear in the correspondence equations (in fact, this is nothing but taking the quotient of the disjoint sum of graphs by the congruence generated by the equations). A resulting glued node/arrow is coloured by green iff at least one of its origins is green. The result of glueing is presented on the Fig.5 and can be denoted by  $\overline{S_I} = (S_1 \oplus S_2 \oplus \overline{S_{CI}})/E_{CI}$  where, as usual, the symbol " / " denotes the operation of taking the quotient. The name *the full integrated sketch* refers to the fact that this sketch contains all the necessary information and, in addition, some derived (green) information.

Actually, while merging the sketch graphs can be performed in a fully automatic way, converting the merge graph into a sketch (by hanging markers on the corresponding diagrams) needs human assistance (and, in general, is undecidable - we will briefly discuss these questions further at section ??).

To end the integration DBA must choose a subsketch  $S_I$  of the full sketch s.t. all items of  $\overline{S_I}$  not contained in  $S_I$  can be extracted from the latter by means of operations from some prescribed list. We will call such subsketches *generating* (Fig. 6)

Note, the derived (green) node 'Topic' cannot be removed since it is the source of the basic (black) arrow 'area'. This implies imposing a special equational constraint on the integrated sketch written down on Fig.6. <sup>4</sup>

Certainly, there are different possibilities of choosing generating subsketches and we will discuss this point in more detail at section 4.

<sup>4</sup>For exposition purposes, in contradistinction to the survey [2] we did not assume that the the class 'Keyword' has the attribute 'Area' and then this attribute must be assigned to the class 'Topic' immediately. This explains the difference between upper halves of our integrated schema and that one obtained in the survey. However, the difference in lower halves (as is shown on Fig.7 needs explanation since our assumptions on semantics behind this fragment of reality were the same.

Actually, the integration in the survey is erroneous. Indeed, according to the right ER-diagram, the attribute 'address' is defined for publishers of all kinds of publications, not only books, but note that the second view does not assume the attribute address for publishers, while in the first view there are no publishers of publications other than books.

Amusingly that we spent a lot of time in thinking why our formal integration algorithm gives incorrect result

## 4 Discussion: mathematical aspects of schema integration

### 4.1 Graph-based algebras.

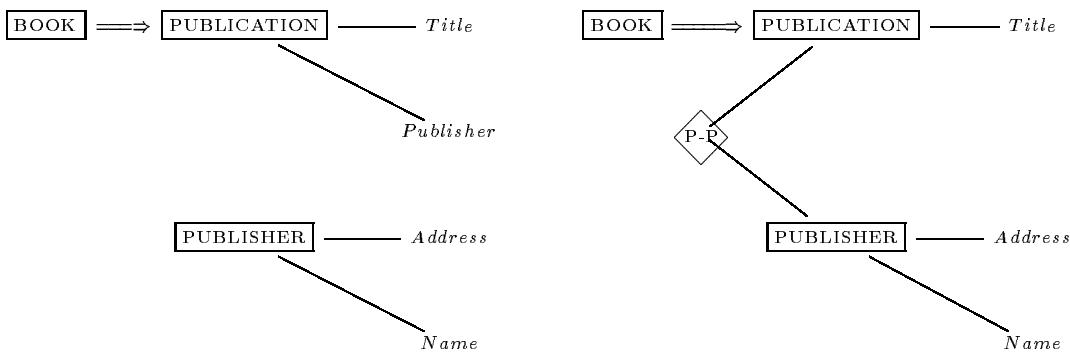
The simple example just described clearly demonstrates that schema integration leads, in general, to database specifications (schemas) consisting of a structural component (sketch, ER-diagram *etc*) coupled with a collection of equational constraints expressing equality of certain queries against the structural schema. We will call constraints of such a type *equational query dependencies (eqd)*. Speaking algebraically, such specifications are nothing but the well known way of defining algebras by generators and defining relations. However, while in the ordinary universal algebra one deals with operations over elements and generating collections of elements, here we deal with generating collections of sets and functions (specified by graphs carrying sketches) and operations over diagrams of sets and functions (specified by diagram markers in sketches).

We see that in contrast to the classical relational algebraic treatment of databases, semantic issues like schema integration naturally lead to considering finitely presentable (fp) *graphical algebras (of sets and functions)*, and we believe that a lot of DB-theoretical problems could be properly stated and examined in this framework.

The subject of fp-algebras is an established focus of universal algebra (see, *eg*, standard textbooks by P.Cohn or G.Grätzer) and category theory as well (see [25] for a modern survey), and large bodies of results are known in both disciplines. However, as a whole they cannot be utilized for DB-theory purposes in an immediate way. Indeed, the universal algebra technique is not suitable since the subject needs graph-based algebraic machinery. On the other hand, the standard categorical framework of finitary monads over locally fp categories is not suitable due to internalization of too many constructs, including signatures of operation symbols and their arities: as we have seen, the subject needs another framework where algebra carriers and operation arities are internalized (*ie*, considered fp-objects in an arbitrary base category) while the very signatures are external sets (maybe, with an additional structure) of operation symbols and should not be considered as objects of the base category.

Thus, what we really need is a graph-based counterpart of the ordinary universal algebra. The main difficulty here is a proper construction of composing diagram operations whose arities are couples of graphical schemas specifying complex data structures. A framework of basic definitions and statements (including sketch parsing) will be briefly outlined at section 5 (see also [13]) and this is the main technical contribution of the paper.

**Remark.** Several attempts to develop a graph-based algebraic machinery were made in category theory (see [34, 25] for a survey). However, all of them deal with a fixed small collection of standard diagram markers denoting universal constructs sufficient to constitute toposes. In contrast, for semantic modeling issues like heterogeneous schema integration one needs to deal with arbitrary collections of diagram markers and operations (very much similar to arbitrary vocabularies in mathematical model theory). In this respect the approach proposed by Makkai [20] is more suitable. However, Makkai's paper addresses to completeness theorems in the classical sense, whereas integration focuses on query operations and *query completeness*, *ie*, definability in the sense of Beth (see [9]). That is why we use the algebraic framework of diagram operations and graphical al-



AGO-integrated ER-diagram (fragment)

Integrated ER-diagram in the survey [2]

Figure 7: Comparison of the results of integration

gebras whenever is possible whereas Makkai works with diagram predicates and diagram inference (cone injectivity); in other words, we are interested in algebra of proofs while Makkai – in provability.

Another limitation of usual categorical approaches to data modeling is that for category theory it is standard to treat subset relationships via monic arrows which is not relevant. Actually, ISA-arrows constitute a distinguished subcategory (in fact, a dominion in the sense of Robinson and Rosolini) and to consider them just monics is actually to sidestep the problem.

Thus, development of graph-based logic and algebra suitable for DB applications needs a special elaboration and actually amounts to a new research direction entitled in [9] by ”Algebraic graph-based model theory for computer science” (see also [8]) which is a graphical and algebraic counterpart of the direction displayed by Makowsky in [21].

**4.2 Metatheory of semantic modeling and schema integration.** Another algebraic issue whose utility can be inferred from the example is the construct of closure  $\mathbf{der}S$  of a given schema (sketch, ER-diagram *etc*)  $S$ . As is well known, the same data can be structured in different yet equivalent ways. Moreover, if speak about heterogeneous schema integration the issue appears extremely actual and occurs in various ways. First of all, one can choose different ways of extending initial sketches to state their correspondence (in this respect our example is not characteristic). Then one would get another full integrated sketch (FIS), and, correspondingly, get another generating subsketches. Thus, various choices of equational specifications of the same correspondence information can lead to different FISs  $\overline{S_I}$ ,  $\overline{S_I''}$ ,  $\overline{S_I'''}$  *etc* and, in addition, there can be different generating subsketches of the same FIS,  $\overline{S_{I^*}}$ ,  $\overline{S_{I^{**}}}$ ,  $\overline{S_{I^{***}}}$  *etc*. However, all these sketches are equivalent in the sense of equality of their closures under *all* derived information that could be extracted from them with the prescribed set of diagram operations (queries). To speak and reason about such questions, it is convenient to introduce a closure operator  $\mathbf{der}$  acting on schemas (sketches): given a schema  $S$ ,  $\mathbf{der}S$  is the (monstrous) closure of  $S$  containing all possible derived items (whose extensions can be extracted from the extensions of  $S$ -items by queries). Then equivalent information capacity of schemas listed above means that  $\mathbf{der}\overline{S_I} = \mathbf{der}\overline{S_I''} = \mathbf{der}\overline{S_I'''} = \mathbf{der}\overline{S_I''''}$  *etc* and, for any such  $\overline{S_I}$ ,  $\mathbf{der}\overline{S_I} = \mathbf{der}\overline{S_{I^*}} = \mathbf{der}\overline{S_{I^{**}}} = \mathbf{der}\overline{S_{I^{***}}}$  *etc*.

Certainly, for particular tasks, such as a concrete task of view integration, we need only finite parts of  $\mathbf{der}S$ , however, the concept of the full closure is very useful.<sup>5</sup>

In precise terms what we are speaking about can be formulated as setting a monad  $\mathbf{der}$  over some category of schemas  $\mathbf{Schema}$ . This constitutes a polymorphic specification machinery when data model is a parameter: for  $\mathbf{Schema}$  one can substitute collections of relational, ER, extended ER, sketch-based *etc* schemas and for  $\mathbf{der}$  some appropriate query mechanism (relational algebra, a kind of ER-algebra (see, *eg*, [23]), sketch algebra *etc*). This gives rise to a precise metatheory of semantic modeling and schema integration constituting a natural framework to handle heterogeneous interoperation issues.

It is characteristic in this respect that such an abstract theory was invoked at a conference recently organized by practitioners ([16]) where it was underlined that ”... *one of the most difficult problems in developing integrated systems is in figuring out what schemas, data and application code mean*”. We will address to the issue of stating basic framework of such a metatheory in section 7

## 5 Diagram operations over a sketch

Operations over sets are specified by pointing their arities, *eg*, *successor* is an unary operation over the set of integers, and *natural join* is a binary operation over the domain of relations on some predefined list of basic domains. For the many sorted case, an arity is not a number but a list of source sorts and a target sort, *eg*, the *pop*-operation over stacks of integers is specified by the arity (‘Int’, ‘Stacks-of-Int’), and the arity of a projection operation over relations is, for example,  $(\text{Rel}_{XYZ}; \text{Rel}_{XY})$ .

However, as we have seen, schema integration needs operations over diagrams (of sets and functions). So, the question arises what is an arity of a diagram operation, and, in general, how to specify compositions of such operations.

Below we present foundations of a formal specification framework for diagram operations over sketches and related machinery in a extent sufficient to explain *sketch parsing* – the main syntactical component of AGO (mentioned at section 3).

<sup>5</sup>This is quite similar to the usefulness of the notion of the set of natural numbers which is nothing but  $\mathbf{der}\{0\}$  for the signature of operations consisting of a single operation *successor*

Let  $\mathbf{G}$  be a finite presheaf topos, *ie*, a functor category of the kind  $\mathbf{Set}^S$  with  $S$  a finite category. As is known,  $\mathbf{G}$  is locally finite presentable (lfp) and, moreover, fp-objects are exactly the finite ones, that is, those functors  $F: S \rightarrow \mathbf{Set}$  for which  $F(i)$  is finite for all  $i \in \text{Obj}S$  (see [20] for basic definitions and results about lfp categories required for our presentation). Objects of  $\mathbf{G}$  should be thought of as (directed multi)graphs, or 2-graphs, or polygraphs or something like of the kind. Further we will interchangeably use terms ‘object’ and ‘graph’ and one can safely think of them as graph-like objects like mentioned above.

Given a lfp category  $\mathbf{C}$ , the collection of all fp-objects will be denoted by  $FP(\mathbf{C})$ .

**5.1 Definition.** (i) A (*fnitary*) *sketch type*  $\tau$  over  $\mathbf{G}$  is defined to be a finite collection  $M$  of *markers*, each assigned with its *arity shape* which is defined to be a finite graph (that is, an fp-object of  $\mathbf{G}$ ) or a recursively defined collection of such objects.

The arity  $\text{Ar}(m)$  of a marker  $m \in M$  describes the kind of diagrams on which the marker can be hung (*eg*, the relationship arc-marker introduced in section 2 can be hung on an arrow source, the identity function marker can be hung on an arrow whose domain and codomain coincide *etc*, see Tables 1,2). Thus,  $\tau = (M, \text{Ar})$  where  $\text{Ar}: M \rightarrow FP(\mathbf{G})$ .

(ii) If  $\tau$  is a type, a  $\tau$ -*sketch* is defined to be a couple  $S = (G, \mathcal{D})$  with  $G$  a graph (*ie*, an object of  $\mathbf{G}$ ) and  $\mathcal{D} = (\mathcal{D}_m, m \in M)$  a family indexed by markers where each  $\mathcal{D}_m$  is a collection (maybe, empty) of diagrams over  $G$  (to be thought of as *marked* by  $m$ ). Certainly, all diagrams in  $\mathcal{D}_m$  have to satisfy the arity condition of  $m$ . For a sketch  $S$  its carrying graph will be denoted by  $|S|$ .

(iii) Given two  $\tau$  sketches  $S, S'$ , a *sketch morphism* from  $S$  to  $S'$  is defined to be a graph morphism  $h: G \rightarrow G'$  s.t. for any diagram  $D \in \mathcal{D}_m$  its image  $h(D) \in \mathcal{D}'_m$  <sup>6</sup> for every  $m \in M$ . This constitutes the category  $\mathbf{Sketch}_\tau(\mathbf{G})$  which is again a finite topos (see [20] for a proof).

A sketch morphism is called *inclusion* if the underlying graph morphism is an inclusion. A sketch  $S'$  is defined to be a *subsketch* of  $S$  if there is an inclusion  $: S' \hookrightarrow S$ .

We will often called  $\tau$ -sketches simply  $M$ -sketches having the arity assignment in mind.

**5.2. Markers** are constraints imposed on diagrams of sets and functions, *ie*, can be considered as diagram predicates. Among them there are operation-determined predicates expressing that certain (output) sets and functions are obtained by application of a corresponding procedure to input sets and functions (see Table 2). For example, labeling a square diagram by the pull-back marker (PB) means that the pair of arrows with the common source was obtained from the pair of arrows with the common target by a certain procedure called (PB) (whose semantics is described in the Table 2). This is quite similar to introducing, *eg*, a ternary predicate  $\langle + \rangle$  s.t.  $\langle + \rangle(x, y, z)$  means that  $z = x + y$ . In addition, it can occur that input data for some operation are specified by a diagram of sets and functions subjected to certain conditions. For example, input data for the operation providing the universal arrow of push-outs, (UAPO in Table 2) are specified by two arrow squares and one of them should be push-out while the other should be commutative.

Thus, we assume that our collection of markers is partitioned as  $M = P \cup O$  with  $P$  a set of predicate (constraint) markers and  $O$  a set of *operation markers* s.t. for

<sup>6</sup>For the formal understanding diagrams as morphisms  $\delta: D \rightarrow G$  where  $D$  is the *shape* of a diagram,  $h(D)$  denotes the composition  $\delta; h$  which is nothing but a diagram over  $G'$  with the same shape  $D$ .

each  $m \in P$  its arity  $\text{Ar}'(m)$  is an object of  $\mathbf{G}$  whereas for each  $m \in O$  its arity  $\text{Ar}(m)$  is a  $\pi$ -sketch, *ie*, an fp-object of  $\mathbf{Sketch}_\pi(\mathbf{G})$  where  $\pi = (P, \text{Ar}')$ . In addition, for every  $m \in O$  a designated subsketch  $\text{Ar}^{\text{in}}(m) \subset \text{Ar}(m)$  is pointed. The complement object  $\text{Ar}(m) \setminus \text{Ar}^{\text{in}}(m)$  will be denoted by  $\text{Ar}^{\text{out}}(m)$ . Note,  $\text{Ar}^{\text{in}}(m) \cap \text{Ar}^{\text{out}}(m) \neq \emptyset$  in general.

Denotational semantics is as follows: for a diagram operation marker  $m$ , as soon as data structured according  $\text{Ar}^{\text{in}}(m)$  are given then data structured according  $\text{Ar}^{\text{out}}(m)$  can be somehow computed. In other words, the arity of an operation marker can be considered as a sketch-based specification of the corresponding procedure.

Correspondingly, an  $O$ -*algebra* is defined to be a  $P$ -sketch  $A$  s.t. for any  $m \in O$  and any  $P$ -sketch morphism

$$e: \text{Ar}^{\text{in}}(m) \rightarrow A$$

(to be thought of as an environment) there is a unique sketch morphism

$$\bar{e}: \text{Ar}(m) \rightarrow A \text{ s.t. } \bar{e} \upharpoonright \text{Ar}^{\text{in}}(m) = e$$

where  $\upharpoonright$  denotes the operation of function restriction (the condition guarantees projection of the input data). Thus, with every marker there is correlated an operation

$$m^A: \mathbf{Sketch}_\pi(\text{Ar}^{\text{in}}(m), A) \rightarrow \mathbf{Sketch}_\pi(\text{Ar}^{\text{out}}(m), A)$$

subjected to the condition above. Clearly, each  $O$ -algebra is a  $\tau$ -sketch but not the reverse.

*Remark* Speaking in general, we should consider the situation when the signature of markers is ordered in a certain way, and  $\text{Ar}^{\text{in}}(m)$  is a sketch of the type consisting of markers preceding  $m$ , their collection is denoted by  $m^- \stackrel{\text{def}}{=} \{n \in M | n \prec m\}$ . For simplicity, in the present paper we restrict ourselves with a very simple situation when  $M = P \cup O$ , for any  $m \in P$  one has  $m^- = \emptyset$  and  $m^- \subset P$  for any  $m \in O$ . The general case will be considered in a forthcoming paper.

The main difficulty in handling diagram operations consists in the diversity of ways they can be composed because inputs and outputs of such operations are complex data structures (specified by sketches). So, it is crucial for the approach to find a proper construction of composing diagram operations, or, in other words, of a term freely composed from diagram operations.

In the ordinary universal algebra, a term can be presented as a tree in a familiar way regardless to its syntactical presentation (with or without brackets, in prefix, suffix or infix form *etc*). So, the key point in handling diagram operation algebras is to design a proper abstract notion of a composed diagram operation (term) independent on its possible syntactical representations.

An appropriate solution was described in [14] where a specification mechanism treating diagram terms as labeled trees was proposed (in a close analogy to the tree presentation of composed operations in universal algebra). The machinery will be elaborated in detail in the forthcoming paper while in the present one we will just outline briefly the basic framework to explain *sketch parsing*.

### 5.3 Construction.

A marker  $c \in O$  is called a *constant* if  $\text{Ar}^{\text{in}}(c) = \emptyset$ .

Each  $P$ -sketch  $X \in FP(\mathbf{Sketch}_\pi(\mathbf{G}))$  determines a constant  $c_X$  with  $\text{Ar}(c_X) = X$  and  $\text{Ar}^{\text{in}}(c_X) = \emptyset$ .

To set a pool of variables we fix a *pool* object  $V \in \mathbf{G}$  (of names) s.t. any finite object of  $\mathbf{G}$  can be embedded into  $V$ . With any finite subobject  $X$  of  $V$  we connect the constant operation marker  $v_X$  with  $\text{Ar}(v_X) = X$ .

Let  $O_V^+$  denotes  $O \cup \{v_X \mid X \text{ is a finite subobject of } V\}$

**5.4 Definition.** Let  $\tau = (P \cup O, \text{Ar}, \text{Ar}^{\text{in}})$  be a sketch type and  $V$  be a pool (of names).

A  $(\tau, V)$ -tree is defined to be a finite tree  $T$  whose nodes are labeled by pairs of the form  $(m, \delta)$  with  $m \in O_V^+$  and  $\delta: |\text{Ar}(m)| \rightarrow V$  a  $\mathbf{G}$ -morphism.

The pair labeling a node  $p$  will be denoted by  $(m_p, \delta_p)$ ;  $\delta_p^{\text{in}}$  and  $\delta_p^{\text{out}}$  denote  $\delta_p \upharpoonright_{\text{Ar}^{\text{in}}(m_p)}$  and  $\delta_p \upharpoonright_{\text{Ar}^{\text{out}}(m_p)}$  respectively;  $V_p$ ,  $V_p^{\text{in}}$  and  $V_p^{\text{out}}$  denote  $\text{Im}(\delta_p)$ ,  $\text{Im}(\delta_p^{\text{in}})$  and  $\text{Im}(\delta_p^{\text{out}})$  respectively;  $p^-$  denotes the set of  $p$ -precessors.

Actually, each node of a tree denotes a computation (query) whose input should be taken from outputs of computations denoted by precessors nodes. The very data transferring is provided by common items (names) in the ranges of the corresponding diagrams.

**5.5 Definition.** (i) A  $(\tau, V)$ -tree is called *well-formed* if for each node  $p$  the following condition is fulfilled:

$$(\text{wft}) \quad V_p^{\text{in}} \subset \bigcup_V \{V_j \mid j \in p^-\}$$

where  $\bigcup_V$  denotes union in the lattice of  $V$ -subobjects.

(ii) A pair of *different* nodes  $(p, q)$  is called *independent* if  $V_p \cap V_q = V_p^{\text{in}} \cap V_q^{\text{in}}$ . Otherwise nodes are called *dependent* and we write  $\text{Dep}(p, q)$ .

Informally, dependency means that some output items of the computation denoted by  $p$  should be always equal to certain output items of the  $q$ -computation and, hence, one has an equational constraint.

(iii) Node  $p$  is called *free* if

(a) it is independent on other nodes, (b)  $\text{CoIm}(V_p^{\text{in}}) = \text{Ar}^{\text{in}}(m_p)$  and (c)  $\delta_p^{\text{out}}$  is monic in  $\mathbf{G}$ .

A tree is called *operational* if its root is a free node.

A tree is called *assertional* if its root is not free.

A tree is called *free* if all its nodes are free.

Evidently, any  $(\tau, V)$ -tree  $T$  determines a  $\tau$ -sketch  $\text{Ske}(T)$  whose carrier graph  $V(T)$  is the union of all  $V_p$  when  $p$  ranges over  $T$ -nodes and, for each  $m \in M$ ,  $\mathcal{D}_m = \{\delta_p \mid m_p = m, p \text{ is a } T\text{-node}\}$ .

In addition, the carrier graph contains a distinguished subobject  $\text{In}(T) \stackrel{\text{def}}{=} \bigcup \{V_p \mid m_p \in O_V^+ \setminus O\}$  specifying input data for the procedure denoted by the entire tree.

**5.6 Definition.** Given a signature (type) of markers,  $\tau$ , a  $\tau$ -specification is a pair  $\mathbf{S} = (S, X)$  with  $S$  a finitary  $\tau$ -sketch and  $X$  a distinguished subobject of its carrier,  $X \subset |S|$ .  $X$  is called *the input object* of  $\mathbf{S}$  and can be endowed with a  $\pi$ -sketch structure in a unique way by coupling it with those  $S$ -diagrams marked by  $P$ -markers whose images are subsets of  $X$ .

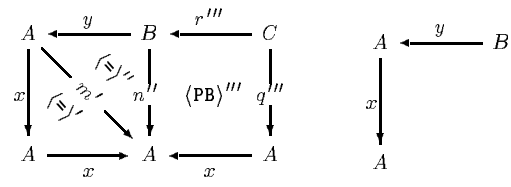
$\mathbf{S}$  is called *well-formed* (wf) if there is a  $(\tau, V)$ -tree  $T$  s.t.  $S = \text{Ske}(T)$  and  $X = \text{In}(T)$ .

**5.7 Proposition.** (*Sketch parsing*).

(i) There is an algorithm either extracting from any given specification  $\mathbf{S} = (S, X)$  a tree  $T = \text{Tree}(\mathbf{S})$  s.t.  $S = \text{Ske}(T)$  and  $X = \text{In}(T)$ , or terminated with failure proving that  $\mathbf{S}$  is not wf.

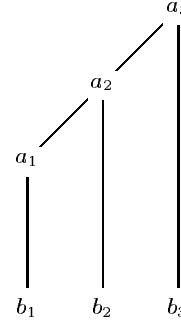
**5.8 Corollary.** The question of whether a given  $\tau$ -specification is well-formed is decidable.

A simple example of sketch parsing is presented on Figure 2.



The sketch of  $\mathbf{S}$

The input object of  $\mathbf{S}$



The tree extracted from  $\mathbf{S}$

Node	Marker	Diagram
$b_1$	$v_{\downarrow \leftarrow}$	$\begin{array}{ccc} A & \xleftarrow{x} & A \\ x \downarrow & & \\ A & & \end{array}$
$b_2$	$v_{\rightarrow}$	$\begin{array}{ccc} B & \xrightarrow{y} & A \end{array}$
$b_3$	$v_{\rightarrow}$	$\begin{array}{ccc} A & \xrightarrow{x} & A \end{array}$
$a_1$	$\langle \equiv \rangle$	$\begin{array}{ccc} A & \xleftarrow{x} & A \\ x \downarrow & \searrow m'' & \\ A & & \end{array}$
$a_2$	$\langle \equiv \rangle$	$\begin{array}{ccc} & & A \\ & \nearrow m' & \uparrow y \\ A & \xleftarrow{n''} & B \end{array}$
$a_3$	$\langle \text{PB} \rangle$	$\begin{array}{ccc} A & \xleftarrow{n''} & B \\ x \uparrow & & \uparrow r''' \\ A & \xleftarrow{q'''} & A \end{array}$

Figure 8: Example of parsing of a  $(\langle \equiv \rangle, \langle \text{PB} \rangle)$ -specification  $\mathbf{S}$

**5.9 Proposition.** Let  $\mathbf{S} = (S, X)$  be a wfs and  $e: X \rightarrow |A|$  be a graph morphism for some  $\tau$ -sketch  $A$  which is actually an  $O$ -algebra. If  $e$  is actually a  $P$ -sketch morphism then either there is no expansion of  $e$  to  $\bar{e}: S \rightarrow A$ : **Sketch $_{\tau}(\mathbf{G})$**  s.t.  $\bar{e} \upharpoonright_X = e$  or there is an exactly one such an expansion.

In the latter case  $(A, e)$  is called a *model* of  $(S, X)$ .

In addition, if  $\text{Tree}(\mathbf{S})$  is free the second alternative always takes place.

## 6 Resolving marker conflicts

Mappings  $\sigma_i$  of graphs underlying local sketches into the global graph carry diagrams over  $G_i$  into diagrams over  $\overline{G_I}$  with the same shape. So, there can well emerge the situation when the same diagram  $D$  in  $\overline{G_I}$ , is the image of several labeled diagrams from local sketches, say,  $D = \sigma_i(D') = \sigma_j(D'')$  for some diagram  $D'$  in  $\overline{S_i}$  labeled by a marker  $m'$  and some diagram  $D''$  in  $\overline{S_j}$  labeled by a marker  $m''$ .

There are three possibilities.

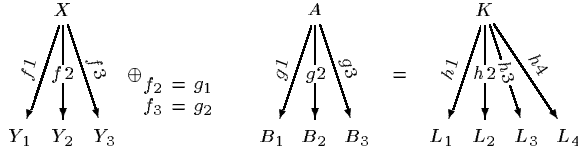
- $m' = m''$ : there is no marker conflict between diagrams,  $D$  is labeled by  $m'$ ;
- $m' \neq m''$  but there is a greatest lower bound (g.l.b.)  $m = m' \wedge m''$ : this is a surmountable marker conflict and  $D$  is labeled by  $m$ ;
- $m' \neq m''$  and there is no (g.l.b.) of  $m$  and  $m'$  or, formally,  $m' \wedge m'' = \perp$  - the *universally unsatisfiable*

*constraint* : this can be qualified as a insurmountable marker conflict which points to inconsistency of views.

After resolving marker conflicts, the graph  $\overline{G_I}$  can be consistently labeled and converted into the *full integrated sketch*. Note, all the mappings  $\sigma_i, i = 1, \dots, n + 1$  will become *sketch morphisms*, ie, mappings of sketches.

Analysis of even simplest constraint conflicts carried out in [Bi86] for the case of functional and inclusion dependencies shows undecidability of the problem of detecting surmountability of conflict between views. As is was said, marker conflicts are the sketch counterpart of the well known in relational data model dependency problem.

Now we present two simple examples of resolving marker conflicts. Let in the first view we have a ternary source  $f_i: X \rightarrow Y_i, i = 1, \dots, 3$  with keys  $[f_1, f_2], [f_2, f_3]$ , and in the second view - a separating source  $g: A \rightarrow B (i=1,2,3)$ , ie, there are no keys other then the whole source) (see Figure 3 below), and correspondence equations amounts to the set  $E_{C_I} = \{f_2 = g_1, f_3 = g_2\}$ . Then the integration of graphs looks as follows:



with mappings  $\sigma_i, i = 1, 2$  defined by as follows:

$S_1$	$f_1$	$f_2$	$f_3$
$\overline{S_I}$	$h_1$	$h_2$	$h_3$

$S_2$	$g_1$	$g_2$	$g_3$
$\overline{S_I}$	$h_2$	$h_3$	$h_4$

To convert the integrated graph into a sketch one must resolve the conflict between markers to be hang on the couple  $[h_2, h_3]$ : according the first view this is a key while the second view does not constrain it. The resolution is evident - the first view constraints is greater (more restrictive) and hence  $[h_2, h_3]$  should be a key, ie, an arc marker must be hang on it. This is an instance of a surmountable marker conflict <sup>7</sup> and the integration looks as presented on Fig.9.

Note, the arc-marker on  $[h_2, h_3, h_4]$  is suppressed since it follows from the marker hung on  $[h_2, h_3]$ .

Another example. Let the marker  $\langle m, n \rangle$  hung on a node bounds the cardinality of a class interpreting the node to be less than  $n$  but greater than  $m$ . Then, clearly,

$$X^{\langle m, n, n \rangle} \oplus_{X=A} A^{\langle m, n, n \rangle} = K^{\langle m, n, n \rangle \wedge \langle m, n, n \rangle},$$

$$\boxed{S_1 \cdot X = S_2 \cdot A = S_I \cdot K}$$

where

$$\langle m, n, n \rangle \wedge \langle m, n, n \rangle = \begin{cases} \langle m, n \rangle & \text{if } m = \max\{m, n\} \leq n = \min\{m, n\} \\ \perp & \text{otherwise} \end{cases}$$

Thus, in the second case we have an insurmountable marker conflict pointing that views are inconsistent.

<sup>7</sup>The situation of resolving conflict during integration as in the example leads to the following important point: not every possible extent of a local view whose marker was increased (view 2 in the example) can be obtained from the corresponding extent of the integrated sketch.

## 7 A category theory approach to metadata modeling

The present section aims at stating a formal framework for data-model-independent (DMI) specification of main data modeling concepts (schema, instance, query etc.) – a crucial issue in handling data interoperation and heterogeneity (cf. the problem italicized in the quotient from [16] at the end of 4.2). We emphasize that what is required here is rather a formal definition of abstract data model capable to support DMI-framework for data modeling than a single unified data model capable to simulate other models. In a sense, this means that we need a framework for *substantial speaking about nothing*.

To approach the task, we assume, first of all, that a collection of abstract objects called *schemas* (one can think of database schemas, or ER-diagrams or something else of the kind) together with another collection of *schema mappings* or *schema morphisms* are fixed so that with each schema morphism  $f$  there are connected its *source* schema  $\langle f$  and its *target* schema  $f \rangle$ ; the notation  $f: S_1 \rightarrow S_2$  will just mean that  $S_1 = \langle f$  and  $S_2 = f \rangle$ . Schemas can be thought of as collections of items structured in a certain way (the same for all schemas) while schema morphisms as mappings of items of their source schemas into items of their target schemas so that the structure is preserved.

This gives rise to a category **Schema** of *schemas* and *schema mappings*, and this is all that we need to know about database schemas. We mean that our data-model-independent answer to the question 'what is a data schema?' is as follows: 'a data schema is an object (node) of the category **Schema**'.

What is data structured according some schema  $S$ , in other words, a database instance over  $S$ ?

Intuitively, given a schema  $S$ , an instance of a database with this schema (further simply an instance) is a mapping which assigns to each item of the schema its extent (*eg*, to any node which is a class name there is assigned a finite set of OIDs while an arrow is assigned with a reference (attribute) function). A natural way to formalize this description is to introduce a semantic "world" schema  $W$  whose items actually are semantic constructs (like sets of OIDs and functions between them) so that any instance could be presented as a schema morphism  $e: S \rightarrow W$ .

For example, let us think of a schema  $S$  from **Schema** as a graph  $G_S = (NdsG_S, ArrsG_S)$  endowed with some additional labeling  $\lambda$  to capture semantic meanings, and of an arrow from **Schema** as a graph morphism compatible with labels. Then an instance of  $S$  can be thought of as a mapping which assigns sets (of object identifiers, or, of material and conceptual objects in the world) to  $G_S$ -nodes, and functions between these sets to  $G_S$ -arrows, s.t. the intended semantic constraints described by  $\lambda$  are satisfied.

Indeed, if we assume

(i) existence of some (generally speaking, hereditary finite) universe of objects,  $\mathcal{O}$ , from which the object sets described above to be taken, and

(ii) organize the collection of subsets of  $\mathcal{O}$  and functions between them into a "world" graph  $G_{\text{Set}(\mathcal{O})}$  (or simply  $G_{(\mathcal{O})}$ ) whose nodes are subsets of  $\mathcal{O}$  while arrows are functions,

then we could be able to consider an instance  $e$  of  $S$  as a graph mapping  $G_S \rightarrow G_{(\mathcal{O})}$ . Thus,  $S$ -structured data appears as a subset of all graph mappings from  $G_S$  into  $G_{\mathcal{O}}$ : there are mappings which are not  $S$ -instances because the latter must be compatible also with semantic labeling  $\lambda$ .

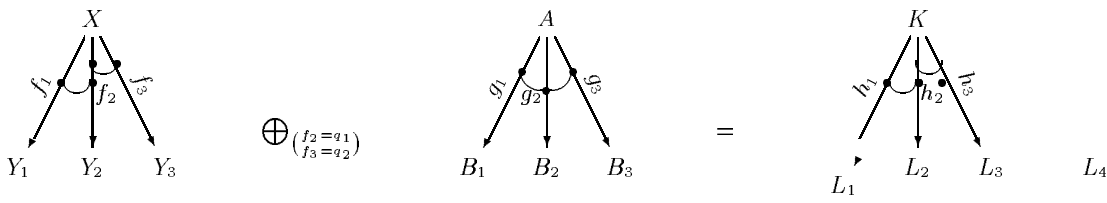


Figure 9: Integration of sketches: resolution of a marker conflict

However, if we will label nodes and arrows of  $G_{(\mathcal{C})}$  in accordance with their real structure thusly converting the *graph*  $G_{\mathcal{C}}$  into a world *schema*  $W$ , then, since schema morphisms are graph morphisms compatible with labeling,  $S$ -instances will prove nothing but schema morphisms from  $S$  into  $W$ . Well, one can argue that this construction is just a way of expressing the ordinary definition of instance, however, it makes possible to develop a data-model-independent framework in a concise way, in particular, to define precisely what semantic overlapping of views does mean, and how it can be specified in schema integration ([5, 11])

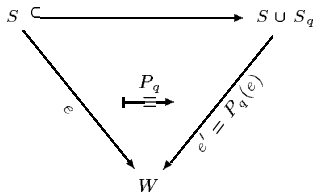
So, we will assume that among objects of **Schema** there is a subcollection **Sem** of schemas regarded as *semantic* so that a database instance over  $S$  is *defined* to be an arrow  $e: S \rightarrow W$  for some  $W \in \mathbf{Sem}$ .

Further, as soon as a schema  $W$  belongs to **Sem**, its items should be thought of as collections of elements, hence, permutations of these elements can transform instances. In our formal framework this can be captured by postulating presence of a new collection of arrows – mappings between instances. To wit, given  $S$  and  $W$ , there are arrows between arrows from  $S$  to  $W$  which can be denoted by  $\pi: e \Rightarrow e' :: S \rightarrow W$ , that is,  $\langle e = \langle e' = S, e \rangle = e' \rangle = W$ ,  $\langle \pi = e, \pi \rangle = e'$ .

These new arrows are of a quite other nature than arrows between schemas, and following the categorical terminology we will call them *2-arrows*. Thus, the underlying graph of **Schema** should be a 2-graph, and since there are defined composition of 2-arrows and identity 2-arrows, actually **Schema** is a 2-category.

What is a query over a schema  $S$ ?

We can assume that a query  $q$  is specified by its schema  $S_q$  together with a mapping  $P_q: \text{inst}(S) \rightarrow \text{inst}(S_q)$ , that is, for any instance  $e: S \rightarrow W$  there is defined another instance  $e' = P_q(e): S_q \rightarrow W$ . Of course, actually  $P_q$  is realized by a computation taking  $e$  as the input and producing  $e'$  as the output. We can add derived items of  $S_q$  to the initial schema so that the query will be specified by the following picture



Thus, a query appears as an operation while any semantic world schema  $W \in \mathbf{Sem}$  as the domain carrying this operation. On the whole, a query mechanism, say, QL, is reduced to a closure operator (or monad)  $\mathbf{der}_{QL}$  over the category **Schema** so that any schema  $S$  can be closed up to a monstrous schema  $\mathbf{der}_{QL} S \supset S$  containing all possible

derived items produced by QL-queries. In addition, semantic schemas turned out to be  $\mathbf{der}_{QL}$ -closed in a sense, *ie*, they are endowed with mappings  $\mu_W: \mathbf{der}_{QL} W \rightarrow W$  corresponding to computing extent of derived items. Then any instance  $e: S \rightarrow W$  can be extended to

$$\bar{e} \stackrel{\text{def}}{=} \mathbf{der}_{QL} e; \mu_W: \mathbf{der}_{QL} S \rightarrow \mathbf{der}_{QL} W \xrightarrow{\mu_W} W.$$

Thus, if an item  $X$  of  $\mathbf{der} S$  is basic, *ie*, belongs to  $S$ , then  $\bar{e}(X) = e(X)$  and the extent of this item is stored in the database, while if  $X$  is derived, *ie*, belongs to  $(\mathbf{der}_{QL} S \setminus S)$  then  $\bar{e}(X)$  is produced by the corresponding query operation or a composition of such operations, *ie*, should be computed.

A query is specified by a subschema  $S_q$  of  $\mathbf{der}_{QL} S$  and the answer is the image of  $S_q$  under the mapping  $\bar{e}$ .<sup>8</sup>

Now we discuss briefly the familiar genericity condition. Informally, it means that, given a query  $q$ , for any

$$\pi: e \Rightarrow e' :: S \rightarrow W$$

one has  $P_q(\pi e) = \pi'(P_q e)$  where  $\pi'$  is the action of the permutation behind  $\pi$  onto  $P_q e$ . Formally, this means that actually  $\mathbf{der}$  is a 2-monad over the 2-category **Schema**, that is, each  $\pi: e \Rightarrow e' :: S \rightarrow W$  gives rise to

$$\bar{\pi} = \mathbf{der} \pi: \bar{e} \Rightarrow \bar{e}' :: \mathbf{der} S \rightarrow \mathbf{der} W.$$

s.t. all necessary commutativity conditions are respected.

Thus, the structural component of an abstract data model is defined to be a triple

$$\mathcal{M} = (\mathbf{Schema}, \mathbf{der}, \mathbf{Sem})$$

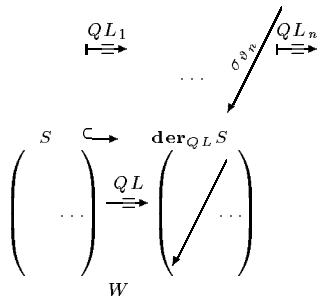
with **Schema** a 2-category,  $\mathbf{der}$  a 2-closure operator (2-monad) over it, and **Sem** a collection of  $\mathbf{der}$ -closed schemas (more accurately,  $\mathbf{der}$ -algebras). Then a database instance is an arrow  $e: S \rightarrow W$  with  $W \in \mathbf{Sem}$  which can be extended in a unique way to the arrow  $\bar{e}: \mathbf{der} S \rightarrow W$  such that the restriction of  $\bar{e}$  on  $S$  equals  $e$ . This can be specified by the following diagram:

$$\begin{array}{ccc} S & \hookrightarrow & \mathbf{der}_{QL} S \\ \left( \begin{array}{c} \vdots \\ \dots \\ \vdots \end{array} \right) & \xrightarrow{QL} & \left( \begin{array}{c} \vdots \\ \dots \\ \vdots \end{array} \right) \\ W & & W \end{array}$$

Now, a (*local*) *view* (over a *global* schema  $S$ ) is naturally defined as a schema morphism  $\sigma_v: S_v \rightarrow \mathbf{der}_{QL} S$ , and it is easy to see that its virtual extent is nothing but a composition

<sup>8</sup> Categorical framework allows to make notions of subobject and image precise.

$$S_1 \xrightarrow{\sigma_1} \text{der}_{QL_1} S_1 \dots S_n \xrightarrow{\sigma_n} \text{der}_{QL_n} S_n$$



$S_i$  and  $QL_i$  are the schema and the query language of the  $i$ th view  
 $\sigma_i : S_i \rightarrow \text{der} S_i, i = 1, \dots, n.$

Figure 10: Meta-schema of a centralized DB with a system of views

$\sigma_v; \bar{e} : S_v \rightarrow W$ . Morphisms of the kind  $S_1 \rightarrow \text{der} S_2$  are well known in category theory under the name of *Kleisly morphisms*, they were carefully examined. So, from the category theory view point a *database view* is nothing but a Kleisly morphism of the monad specifying query mechanism.

On the whole, a centralized DB-system with a system of views over it can be specified by the diagram presented on Figure 10. Note, the corresponding graphical image is not merely a picture convenient for heuristic discussion but also a precise formal specification. A similar meta-schema can be built to represent a general federated DB architecture (substantially described in [27]).

The framework just outlined is close to the institution theory framework familiar in the specification-languages-in-programming community (see Goguen and Burstall [17]). The essential distinction is in setting a monad over the category of schemas (signatures in the institution theory terminology; see discussion in [15]). On the other hand, the framework is a categorial generalization of classical algebraic logic going back to Tarski and Halmos (see Plotkin [24] for exposition and DB applications).

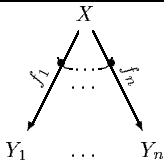
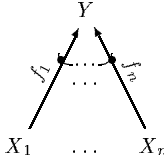
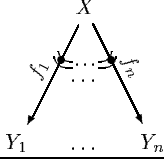
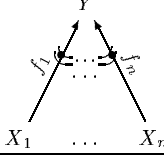
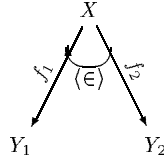
A few basic definitions and results stating when specifications can be replaced by (Lindenbaum-Tarski) algebras in a very general context were proved in [12]

## References

- [1] M. Barr and C. Wells. *Category Theory for Computing Science*. Prentice Hall International Series in Computer Science, 1990.
- [2] C. Batini, M. Lenzerini, and S. Navathe. A comparative analysis of methodologies for database schema integration. *ACM Computing Surveys*, 18(4):323–364, 1986.
- [3] P. Buneman, S. Davidson, and A. Kosky. Theoretical aspects of schema merging. In *EDBT'92*, 1992.
- [4] B. Cadish and Z. Diskin. Algebraic Graph-Oriented = Category Theory Based. Manifesto of categorizing database theory. Technical Report 06, Frame Inform Systems, Riga, Latvia, 1994. (On ftp: //ftp.cs.chalmers.se/pub/users/diskin/manifest.\*).
- [5] Cadish and Z. Diskin. Algebraic graph-oriented approach to view integration. Part I: Specification framework and general strategy. Technical Report 01, Frame Inform Systems, Riga, Latvia, 1993.

- [6] B. Cadish and Z. Diskin. Algebraic graph-based approach to management of multibase systems, I: Schema integration via sketches and equations. In *Next Generation of Information Technologies and Systems, NGITS'95*, 2nd Int. Workshop, Naharia (Israel) June 1995, 1995. Extended abstract is available by ftp: //ftp.cs.chalmers.se/pub/users/diskin/integr.\*).
- [7] U. Dayal and H. Hwang. View definition and generalization for database integration of a multibase system. *IEEE Trans. Software Eng.*, 10(6):628–644, 1984.
- [8] Z. Diskin. Formalization of graphical schemas: General sketch-based logic vs. heuristic pictures. To appear in Proc. 10th Int. Congress "Logic, Methodology and Philosophy of Science" (Florence, Italy), August 1995 (On ftp: //ftp.cs.chalmers.se/pub/users/diskin/florence.\*).
- [9] Z. Diskin. Towards algebraic graph-based model theory for computer science. Accepted for "Logic Colloquium'95" and will appear in "Bulletin of Symbolic Logic" (On ftp: //ftp.cs.chalmers.se/pub/users/diskin/LC95abstr.\*).
- [10] Z. Diskin. A mathematical framework for comparing expressive powers of different data model. In *Methods of Database Design, MDD'92*, Baltic Int. Conference, pages 28–33, Riga, Latvia, 1992.
- [11] Z. Diskin. Algebraic graph-based approach to management of multibase systems, II: Algebraic aspects of schema integration. Technical Report 07, Frame Inform Systems/LDBD, Riga, Latvia, 1994.
- [12] Z. Diskin. Algebraizing institutions: Incorporating algebraic logic methodology into the institution framework for building specifications. Technical Report 04, Frame Inform Systems/LDBD, Riga, Latvia, 1994.
- [13] Z. Diskin. Formalizing graphical schemas for conceptual modeling: Sketch-based logic vs. heuristic pictures. Technical Report 01, Frame Inform Systems/LDBD, Riga, Latvia, 1995. To appear in Proc. of Int. KRUSE Symp. "Knowledge Retrieval, Use and Storage for Efficiency" (Californy, Santa Krus), August 1995.
- [14] Z. Diskin and I. Beylin. What is an operation over sketches? A talk given at Winter Meeting of Computer Science Department, Chalmers University, Göteborg, January, 1995.
- [15] Z. Diskin and B. Cadish. Abstract queries, schema transformations and algebraic theories: An application of categorial algebra to database theory. In *Acta Universitatis Latviensis. Matematika*, volume 595, pages 83–96. University of Latvia, Riga, Latvia, 1994.
- [16] P. Drew, R. King, D. McLeod, M. Rusinkiewicz, and A. Silberschatz. Report on the workshop on semantic heterogeneity and interoperation in multidatabase systems. *SIGMOD Record*, 22(3):47–56, 1993.
- [17] G.A. Goguen and R.M. Burstall. Institutions: Abstract model theory for specification and programming. *Journal of ACM*, 39(1):95–146, 1992.
- [18] L. Kalinichenko. Methods and tools for equivalent data model mapping construction. In *Advances in Database Technology - EDBT'90*, pages 92–119, 1990.

- [19] J.A. Larson, S.B. Navathe, and R. Elmasri. A theory of attribute equivalence in databases with application to schema integration. *IEEE Trans. Software Eng.*, 15(4), 1989.
- [20] M. Makkai. Generalized sketches as a framework for completeness theorems. Technical report, McGill University, 1994. To appear in *J.Pure Applied Algebra*. (On ftp: [//triples.math.mcgill.ca/pub/makkai/sketch](http://triples.math.mcgill.ca/pub/makkai/sketch)).
- [21] J. Makowsky. Model theory and computer science: An appetizer. In S. Abramsky, D.M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science*, volume 1. Oxford University Press, 1992.
- [22] A. Motro. Superviews: Virtual integration of multiple databases. *IEEE TOSE*, 13(7):785–798, 1987.
- [23] C. Parent and S. Spaccapietra. Complex object modeling: an entity relationship approach. In *Nested relations and complex objects in Databases*, LNCS'361, pages 272–296, 1987.
- [24] B. Plotkin. *Universal algebra, Algebraic logic and Databases*. Kluwer Publ.Co, 1993.
- [25] E. Robinson. Variations on algebra: monadicity and generalizations of equational theories. Technical report, University of Sussex, 1994. On ftp: [//ftp/theory.doc.ic.ac.uk/papers/](http://ftp/theory.doc.ic.ac.uk/papers/).
- [26] M.H. Scholl, H.-J. Schek, and M. Tresch. Object algebra and views for object bases. In *Int.Workshop on Distributed Object Management, Edmonton, Canada*, 1992.
- [27] A. Sneth and C. Larson. Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys*, 1990.
- [28] S. Spaccapietra and C. Parent. Conflicts and correspondence assertions in interoperable databases. *ACM SIGMOD Record*, 20(4):49–54, 1991.
- [29] S. Spaccapietra, C. Parent, and Y. Dupont. Model-independent assertions for integration of heterogeneous schemas. *Very Large Databases Journal*, 1(1), 1992.
- [30] S. Spaccapietra, C. Parent, and Y. Dupont. View integration: a step forward in solving structural conflicts. *IEEE Transactions on KDE*, 1992.
- [31] M. Tsalenko. *Semantic modeling in databases (in Russian)*. Nauka Publ.Co, 1989.
- [32] C. Tuijn. Data modeling from a categorical perspective. Ph.D Thesis, Antwerpen,1994, 1994.
- [33] C. Tuijn and M. Gussens. Views and decompositions from a categorical perspective. In *4th Int.Conf. on Database Theory, ICDT'92*, volume 646 of *Lect.Notes in Com.Sci*, pages 99–112, 1992.
- [34] C. Wells. Sketches: Outline with references. On ftp [//ftp.cwru.edu/math/wells.\\*](http://ftp.cwru.edu/math/wells.*).
- [35] S. Widjojo, R. Hull, and D. Wale. A specification approach to merging persistent object systems. In *4th Int. Workshop on Persistent Object Systems*, 1990.
- [36] K. Yetongnon, M. Andersson, Y. Dupont, S. Spaccapietra, H.-J. Schek, M. Scholl, and M. Tresch. The FEMUS experience. In *Semantics of interoperable database systems, IFIP DS-5*, 1992.

Name	Arity Shape and Designation	Denotational Semantics
Separating Source		$(\forall x, x' \in X) x \neq x' \text{ implies } f_i(x) \neq f_i(x')$ for some $f_i$
Monic Arrow †	$X \xrightarrow{f} Y$	$(\forall x, x' \in X) x \neq x' \text{ implies } f(x) \neq f(x')$
ISA-Arrow or Inclusion	$X \xrightarrow{f} Y$ or $X \hookrightarrow^f Y$	$X \subset Y$ and $f(x) = x$ for all $x \in X$
Covering Flow		$(\forall y \in Y)(\exists i < n)y \in f_i(X_i)$
Cover †	$X \xrightarrow{f} Y$	$Y = f(X)$
Inversion	$X \xrightarrow{f} Y$ $\xleftarrow{g} Y$ $\xrightarrow{(-1)} Y$	$(\forall y \in Y)f(g(y)) = y$
Maximal Separating Source		$(\forall x, x' \in X) x \neq x' \text{ implies } f_i(x) \neq f_i(x')$ for some $f_i$ and $(\forall y_1 \in Y_1, \dots, \forall y_n \in Y_n) \exists x \in X$ s.t. $f_1(x) = y_1, \dots, f_n(x) = y_n$ .
Disjoint Covering Flow		$(\forall y \in Y \exists i < n)y \in f_i(X_i)$ and $i \neq j \text{ implies } f_i(X_i) \cap f_j(X_j) = \emptyset$
$\epsilon$ -relation		$(\forall y, y' \in Y_1) y \neq y' \text{ implies}$ $\{f_2x : x \in f_1^{-1}y\} \neq \{f_2x : x \in f_1^{-1}y'\}$ , i.e., there is an embedding $Y_1 \hookrightarrow \mathbf{Powerset}Y_2$ .

The constructions tagged with † are nothing than trivial cases of previous markers.

Table 1: A collection of diagram constraints

Table 2: A collection of diagram operations

Name	Designation in texts diagrams		Arity Shape		Denotational Semantics
			Input sketch	Output sketch	
Identity	<i>Id</i>	$\langle \text{id} \rangle$	$X$	$X \xrightarrow[\text{f}]{\langle \text{id} \rangle} X$	$(\forall x \in X) f(x) = x$
Glueing	<i>Glue</i>	$\langle ! \rangle$	$X$	$X \xrightarrow[\text{f}]{\langle ! \rangle} U$	$U = \{*\}; (\forall x \in X) f(x) = *$
Image	<i>Im</i>	$\langle \text{Im} \rangle$	$X \xrightarrow[\text{f}]{} Y$	$\begin{array}{ccc} I & \xrightarrow{i} & Y \\ f' \uparrow & \nearrow (\text{Im}) & \nearrow f \\ X & & \end{array}$	$I = \{f(x) : x \in X\}$ $(\forall x \in X) f'(x) = f(x)$
Composition	$;$	$\langle = \rangle$	$\begin{array}{ccc} Z & \xrightarrow{g_2} & Y \\ g_1 \uparrow & & \\ X & & \end{array}$	$\begin{array}{ccc} Z & \xrightarrow{g_2} & Y \\ g_1 \uparrow & \nearrow (\text{=}) & \nearrow f \\ X & & \end{array}$	$(\forall x \in X) f(x) = g_2(g_1(x))$
Equalizer	<i>Eqvr</i>	$\langle \text{Eq} \rangle$	$\begin{array}{ccc} X & \xrightarrow[\text{g}]{\text{f}} & Y \\ \text{+} & & \end{array}$	$E \xrightarrow[e]{\langle \text{Eq} \rangle} X \xrightarrow[\text{g}]{\text{f}} Y$	$E = \{x \in X : fx = gx\}$
Push-Out (of ISA-arrows)	<i>PO</i>	$\langle \text{PO} \rangle$	$\begin{array}{ccc} C & \xrightarrow{\text{f}} & A \\ \parallel & & \\ g \downarrow & & \\ B & & \end{array}$	$\begin{array}{ccccc} C & \xrightarrow{\text{f}} & A & & \\ \parallel & & \parallel & & \\ g \downarrow & & k \downarrow & & \\ B & \xrightarrow{\text{=}} & S & \xrightarrow{l} & A \\ & & \parallel & & \\ & & B & & \end{array}$	$S = \{(a, 1) : a \in (A \setminus C)\} \cup \{(b, 2) : b \in (B \setminus C)\} \cup \{(c, 3) : c \in C\}$
Universal Arrow of PO	<i>UAPO</i>	$\langle ! \rangle$	$\begin{array}{ccc} Y & \xleftarrow{i} & A \\ \omega \swarrow & \searrow & \parallel \\ & S & \leftarrow A \\ & \parallel & \\ & B & \leftarrow C \end{array}$	$\begin{array}{ccc} Y & \xleftarrow{i} & A \\ \omega \swarrow & \searrow & \parallel \\ & S & \leftarrow A \\ & \parallel & \\ & B & \leftarrow C \end{array}$	$(\forall s \in S)$ $u(s) = \begin{cases} i(a) & \text{if } s = (a, 1), \\ j(b) & \text{if } s = (b, 2), \\ i(c) = j(c) & \text{if } s = (c, 3) \end{cases}$
Pull-back	<i>PB</i>	$\langle \text{PB} \rangle$	$\begin{array}{ccc} & A & \\ & \downarrow f & \\ B & \xrightarrow{g} & X \end{array}$	$\begin{array}{ccc} R & \xrightarrow{p} & A \\ q \downarrow & & \downarrow f \\ B & \xrightarrow{g} & X \end{array}$	$R = \{(a, b) \in A \times B : fa = gb\}$ $p, q$ are projections
Coimage	<i>CoIm</i>	$\langle \text{CoIm} \rangle$	$\begin{array}{ccc} & B & \\ & \parallel b & \\ X & \xrightarrow{f} & Y \end{array}$	$\begin{array}{ccc} B' & \xrightarrow{f'} & B \\ b' \parallel & & \parallel b \\ X & \xrightarrow{f} & Y \end{array}$	$B' = \{x \in X : fx \in B\}$ $f' = \text{restriction of } f \text{ on } B'$

Following the tradition, we will often omit the marker of composition. So,  $\langle = \rangle$  can be considered as the default marker. In contrast, the diagrams not assumed to be commutative are marked with puncture sign  $\text{+}$ .