Theory of sketches for database mappings

Zoran Majkić

International Society for Research in Science and Technology,
Tallahassee, FL 32316 – 2464, USA
Email: majk.1234@yahoo.com

Bhanu Prasad*

Department of Computer and Information Sciences,
Florida A&M University,
Tallahassee, Florida 32307, USA
Email: prasad@cis.famu.edu
*Corresponding author

Abstract: This paper presents two basic operations, separation and data federation, for database schemas that are used in database mapping systems and explains why functorial semantics for database mappings need a new base category instead of usual **Set** category. A definition of graph G for a schema database mapping system and the definition of its sketch category $\mathbf{Sch}(G)$ are presented. Based on this framework, functorial semantics for database mapping systems with the new base category **DB** are also presented.

Keywords: relational databases; database mappings; denotational semantics.

Reference to this paper should be made as follows: Majkić, Z. and Prasad, B. (2015) 'Theory of sketches for database mappings', *Int. J. Intelligent Information and Database Systems*, Vol. 9, No. 1, pp.32–53.

Biographical notes: Zoran Majkić received his Master of Telecommunication in Electromagnetic Engineering from ETF University of Belgrade, where he worked for two years as an Assistant Professor. After that he changed his work and started doing research in Computer Science and received his PhD in Computer Science from La Sapienza University, Roma, Italy. He is currently working as an Information Technology Advisor and also doing research in computer science and quantum physics. His research interests include knowledge-base and database systems, artificial intelligence, and category theory and algebras.

Bhanu Prasad received his Master of Technology and PhD, both in Computer Science, from Andhra University and Indian Institute of Technology Madras, respectively. He is currently working as an Associate Professor in the Department of Computer and Information Sciences at Florida A&M University in Tallahassee, Florida, USA. His current research interests include knowledge-based systems and artificial intelligence.

1 Introduction

Most of the work in data integration or exchange and P2P framework, particularly for the integrity constraints, in order to define right models for certain answers, is based on a logical point of view in a 'local' mode (i.e., source-to-target database). However, a general 'global' problem that deals with *composition* of complex partial mappings that involve a number of databases has not been given the right attention in these works.

The current research is an attempt to provide a right solution for the general problem of complex database-mappings and for the high level algebra operators namely separation and data federation, by preserving the traditional logical language for schema database mappings.

A limited amount of research has been reported in the literature (Madhavan et al., 2002; Alagić and Bernstein, 2002; Davidson et al., 1998; Melnik et al., 2003) that addresses the general problem presented in the current research. The research presented in Alagić and Bernstein (2002) uses a category theory and it is too restrictive because its institutions can be applied only for *inclusion* mappings between databases.

A lot of research has been done on sketch-based denotational semantics for databases (Lellahi and Spyratos, 1990; Rosebrugh and Wood, 1992; Diskin and Cadish, 1995; Johnson et al., 2000). However, that research used the elements of an ER-scheme of a database such as relations, attributes, etc. as the objects of a sketch category but not the *whole* database as a single object. Hence, we need a framework of inter-databases mappings. The research presented in Diskin (1997) has shown that to progress towards more expressive sketches w.r.t. the original Ehresmann's sketches for diagrams with limits and coproducts, by eliminating non-database objects, for example, Cartesian products of attributes or powerset objects, we need *more expressive arrows* for sketch categories [i.e., diagram predicates in Diskin (1997) that are analogous to the approach of Makkai in Makkai (1994)]. As we progress towards a more abstract vision in which objects are the whole databases, following the approach of Makkai, we obtain more complex arrows in this new basic category **DB** for databases in which objects are just the database instances (each object is a set of relations that compose this database instance). Such arrows are not just simple functions as in the case of base **Set** category but complex trees (i.e., operads) of view-based mappings. In this way, while Ehresmann's approach prefers to deal with a few fixed diagram properties [commutativity and (co)limitness], we enjoy the possibility of setting a full relational-algebra signature of diagram properties.

The current research is an attempt to provide a proper solution for this problem while preserving the traditional common logical language for the schema database mapping definitions.

The *instance level* base database category **DB** has been introduced for the first time in Majkić (2003a) and it was also used in Majkić (2003b). While general information about categories can be found in classic text books such as Mac Lane (1971), more information about the database category **DB** with set of its objects Ob_{DB} and set of its morphisms Mor_{DB} is available in Majkić (2008). The current research emphasises some of the basic properties of this **DB** category, in order to make the presentation more self-contained.

Every object, denoted by A, B, C,.. of this category is a database instance and it is composed of a set of n-ary relations $a_i \in A$, $i = 1, 2, ...$ and these relations are called 'elements of A'.

We consider the views as a universal property for databases. Views are the possible observations of the information contained in an instance-database and we may use them to establish an equivalence relation between the databases. The power-view operator T, with the domain and codomain equal to the set of all database instances, has been defined such that for any object (database) A the object TA denotes a database composed of the set of *all views* of A (Majkić, 2003a). The object TA, for a given database instance A, corresponds to the quotient-term algebra $\mathcal{L}_A/_\approx$, in which the carrier is a set of equivalence classes of closed terms of a well defined formulae of a relational algebra which is 'constructed' by the following: Σ_R-constructors (i.e., relational operators select, project, join and union, in SPRJU algebra) and symbols (attributes of relations) of a database instance A, and constants of attribute-domains.

Different properties of the base **DB** category were considered in the literature (Majkić, 2009a, 2009b, 2011a, 2011b; Majkić and Prasad, 2010), where the basic power-view operator T is extended to the endofunctor $T : \mathbf{DB} \to \mathbf{DB}$.

The connection between a logical (schema) level and computational category **DB** is based on the *interpretation* functors. Thus, each rule-based conjunctive query at schema level over a database $\mathcal{A}$ is translated (by an interpretation functor) in a morphism in **DB**, from an instance-database A (a model of the database schema $\mathcal{A}$) to the instance-database TA composed by all views of A.

1.1 Basic database concepts

The database mappings, for a given logical language [for default we assume the First-Order Language (FOL)], defined usually at a schema level (π_1 and π_2 denote first and second projections, $\biguplus$ denotes disjoint union, and $\mathcal{N}$ represents the set of natural numbers), are as follows:

- A *database schema* is a pair $\mathcal{A} = (S_A, \Sigma_A)$ where $S_A = \pi_1(\mathcal{A})$ is a countable set of relation symbols $r \in R,\ ar : R \to \mathcal{N}$, with finite arity (finite list of attributes $\mathbf{x} =< x_1, ..., x_n >, n = ar(r) \geq 1$), disjoint from a countable infinite set **att** of attributes (for any single attribute $x \in$ **att**, the domain of x is a non-empty subset $dom(x)$, of a countable set of individual symbols **dom**, disjoint from **att**), such that for any $r \in R$, the sort of R is a finite sequence of elements of **att**.
  $\Sigma_A = \pi_2(\mathcal{A})$ denotes a set of closed formulas (without free variables) called integrity constraints of the sorted First-Order Language (FOL) with sorts **att**, constant symbols **dom**, relational symbols in S_A, and no function symbols.

  We denote the set of all database schemas for a given (also infinite) set R by $\mathbb{S}$.

  We denote the empty database schema (where $\pi_1(\mathcal{A}_\emptyset)$ and $\pi_2(\mathcal{A}_\emptyset)$ are empty sets) by $\mathcal{A}_\emptyset$. A *finite* database schema $\mathcal{A}$ is composed of a finite set S_A, so that the set of all attributes of such a database is finite.

- We consider a rule-based *conjunctive query* over a database schema $\mathcal{A}$ as an expression $q(\mathbf{x}) \longleftarrow R_1(\mathbf{u}_1), ..., R_n(\mathbf{u}_n)$, where $n \geq 0$, R_i are either the relation names (at least one) in $\mathcal{A}$ or the built-in predicates (ex. $\leq, =$, etc..), q is a relation name not in $\mathcal{A}$ and $\mathbf{u}_i$ are free tuples (i.e., may use either variables or constants). Recall that if $\mathbf{v} = (v_1, .., v_m)$ then $R(\mathbf{v})$ is a shorthand for $R(v_1, .., v_m)$. Finally, each variable occurring in $\mathbf{x}$ must also occur at least once in $\mathbf{u}_1, ..., \mathbf{u}_n$.

Rule-based conjunctive queries (called rules) are composed of a subexpression $R_1(\mathbf{u}_1), ..., R_n(\mathbf{u}_n)$ that is the *body* and $q(\mathbf{x})$ that is the *head* of this rule. If we can find values for the variables of the rule, such that the body is logically satisfied, then we can deduce the head-fact. This concept is captured by a notion of 'valuation'. In the rest of this paper, a deduced head-fact is called 'a resulting *view* of a query $q(\mathbf{x})$ defined over a database $\mathcal{A}$', and it is denoted by $\|q(\mathbf{x})\|$. Recall that the conjunctive queries are monotonic and satisfiable and the Yes/No conjunctive queries are the rules with an empty head.

- We consider that a *mapping* between two database schemas $\mathcal{A}$ and $\mathcal{B}$ is expressed by an union of 'conjunctive queries with the same head'. Such mappings are called 'view-based mappings' and can be defined by the set $\mathcal{M} = \{q_{Ai}(\mathbf{x}_i) \Rightarrow q_{Bi}(\mathbf{x}_i) | 1 \leq i \leq n\}$, where $\Rightarrow$ is the logical implication between these conjunctive queries $q_{Ai}(\mathbf{x}_i)$ and $q_{Ai}(\mathbf{x}_i)$, over the databases $\mathcal{A}$ and $\mathcal{B}$ respectively.

  We consider *a view* of an instance-database A an n-ary relation (set of tuples) obtained by a 'select-project-join + union' (SPJRU) query $q(\mathbf{x})$ (it is a term of SPJRU algebra) over A. If this query is a finite term of this algebra then it is called a 'finitary view'. Notice that a finitary view can also have an infinite number of tuples.

- An *instance* of a database $\mathcal{A}$ is given by $A = (\mathcal{A}, I_A)$, where I_A is an Tarski's FOL interpretation function that satisfies *all* integrity constraints in Σ_A and maps each relational symbol of S_A (n-ary predicate in FOL) into an n-ary relation $a_i \in A$ (also called 'element of A'). Thus, a relational instance-database A is a set of n-ary relations and they are managed by relational database systems (RDBMSs).

  Given two autonomous instance-databases A and B, we can make a *federation* of them, i.e., their disjoint union $A \biguplus B$, in order to be able to *compute* the queries with relations of both autonomous instance-databases.

  A federated database system is a type of meta-database management system (DBMS) which transparently integrates multiple autonomous database systems into a single federated database. The constituent databases are interconnected via a computer network, and may be geographically decentralised. Since the constituent database systems remain autonomous, a federated database system is a contrastable alternative to the (sometimes daunting) task of *merging* together several disparate databases. A federated database, or virtual database, is the fully-integrated, logical composite of all constituent databases in a federated database system.

  Sheth and Larson (1990) were among the first to define a federated database system, as one which "define[s] the architecture and interconnect[s] databases that minimise central authority yet support partial sharing and coordination among database systems". Among other surveys, McLeod and Heimbigner (1985) "define a Federated Database as a collection of cooperating component systems which are autonomous and are possibly heterogeneous" (wikipedia page http://en.wikipedia.org/wiki/Federated-database-system).

1.2 Database (DB) category

Based on an observational point of view for relational databases, we may introduce a category **DB** (Majkić, 2008) for instance-databases and view-based mappings between them, with the set of its objects Ob_{DB} and the set of its morphisms Mor_{DB}, such that:

1 Every object (denoted by A, B, C,...) of this category is an instance-database, composed of a set of n-ary relations $a_i \in A$, $i = 1, 2, ...$, also called 'elements of A'. We define a universal database instance Υ as the union of all database instances, i.e., $\Upsilon = \{a_i | a_i \in A, A \in Ob_{DB}\}$. Υ is the top object of this category.

   $\Upsilon = T\Upsilon$ because every view $v \in T\Upsilon$ is an instance-database as well, thus $v \in \Upsilon$. Vice versa, every element $r \in \Upsilon$ is a view of Υ as well, thus $r \in T\Upsilon$.

   Every object (instance-database) A has also the empty relation $\perp$. The object composed of only this empty relation is denoted by $\perp^0$ and hence $T\perp^0 = \perp^0 = \{\perp\}$.

   Two objects A and B are isomorphic in **DB**, denoted by $A \simeq B$, if $TA = TB$.

   For any instance-database A it holds that $A \subseteq TA$ and $A \simeq TA$.

   Any empty database (a database with only empty relations) is isomorphic to this bottom object $\perp^0$.

2 Morphisms of this category are all possible mappings between instance-databases *based on views* and are defined by formalism of operads in the rest of the paper.

In the rest of paper, the objects in **DB** (i.e., instance-databases) are simply called as databases, when it is clear from the context. Each atomic mapping (morphism) in **DB** between two databases is generally composed of three components: the first one corresponds to conjunctive query q_i over a source database that defines this view-based mapping, the second (optional) w_i 'translates' the obtained tuples from domain of the source database (for example in Italian) into terms of domain of the target database (for example in English), and the last component v_i defines the contribution of this mappings to the target relation, i.e., a kind of global-or-local-as-view (GLAV) mapping (sound, complete or exact).

In the rest of this paper we consider a more simple case without the component w_i and we also introduce two functions ∂_0 and ∂_1 such that $\partial_0(q_{A_i}) = \{r_{i1}, ..., r_{ik}\}$ (the set of relations used in the query formula $q_{A_i}(\mathbf{x})$) and $\partial_1(q_{A_i}) = \{r_i\}$, with obtained view $r_i = \|q_{A_i}(\mathbf{x})\|$. Thus, we can formally introduce a theory for view-mapings based on operads:

Definition 1: We define the following two types of basic mappings:

- logic-sentence mapping: For any sentence, i.e., a logic formula φ_i without free variables over a Database schema $\mathcal{A}$, we can define a schema mapping $\varphi_i : \mathcal{A} \longrightarrow \mathcal{A}_\varnothing$. The unique instance-database of the empty shema $\mathcal{A}_\varnothing$ is denoted by $\perp^0 = \{\perp\}$, where $\perp$ denotes the empty relation. Consequently, for each interpretation α, $\alpha^*(\mathcal{A}_\emptyset) = \perp^0$. This kind of schema mappings are used for the integrity constraints over database schemas and for Yes/No queries, as specified in Section 3.

- View-mapping: For any *query* (a logic formula with free variables) over a schema $\mathcal{A}$ we can define a schema map $q_i : \mathcal{A} \longrightarrow \{r_i\}$, where $q_i \in O(r_{i1}, ..., r_{ik}, r_i)$, $Q = (r_{i1}, ..., r_{ik}) \subseteq \mathcal{A}$.

  For a given α the corresponding view-map at instance level is $q_{A_i} = \{\alpha(q_i)$ and $q_\perp\} : A \longrightarrow TA$, with $\perp \in A = \alpha^*(\mathcal{A}) \subseteq TA)$, $\partial_0(q_\perp) = \partial_1(q_\perp) = \{\perp\}$. For simplicity, in the rest of this paper, we drop the component $q_\perp$ of a view-map and assume implicitly such a component; thus, $\partial_0(q_{A_i}) = \alpha^*(Q) \subseteq A$ and $\partial_1(q_{A_i}) = \{\alpha(r_i)\} \subseteq TA$ is a singleton with the unique element equal to view obtained by a 'select-project-join+union' term $\widehat{q_i}$.

Thus, we introduce an *atomic morphism* (mapping) between two databases as a set of simple view-mappings as follows:

Definition 2: Atomic morphism: Every *schema mapping* $f_{Sch} : \mathcal{A} \longrightarrow \mathcal{B}$, based on a set of query-mappings q_i, for a finite natural number N is defined as

$$f_{Sch} \triangleq \{\ v_i \cdot q_i \mid q_i \in O(r_{i1}, ..., r_{ik},\ r_i'),\ v_i \in O(r_i', r_i), \\ \{r_{i1}, ..., r_{ik}\} \subseteq \mathcal{A},\ r_i \in \mathcal{B}, 1 \leq i \leq N\}.$$

Its corresponding *complete morphism* at instance database level is

$$f = \alpha^*(f_{Sch}) \triangleq \{\ q_{A_i} = \alpha(v_i) \cdot \alpha(q_i) \mid\ v_i \cdot q_i \in f_{Sch}\} : A \to B,$$

where each $\alpha(q_i)$ is a query computation, with obtained view $\alpha(r_i') \in TA$ for an instance-database $A = \alpha^*(\mathcal{A}) = \{\alpha(r_k) \mid r_k \in \mathcal{A}\}$ and $B = \alpha^*(\mathcal{B})$.

Let π_{q_i} be a projection function on relations, for all attributes in $\partial_1(\alpha(q_i)) = \{\alpha(r_i'')\}$. Then each $\alpha(v_i) : \alpha(r_i') \longrightarrow \alpha(r_i)$ is one tuple-mapping function, used to distinguish sound and exact assumptions on the views, as follows:

1 *Inclusion* case, i.e., when $\alpha(r_i') \subseteq \pi_{q_i}(\alpha(r_i))$. Then for any tuple $t \in \alpha(r_i')$ there exists $t_1 \in \alpha(r_i)$ such that $\pi_{q_i}(\{t_1\}) = t$ and $\alpha(v_i)(t) = t_1$.

2 *Exact* case, i.e., special inclusion case when $\alpha(r_i') = \pi_{q_i}(\alpha(r_i))$.

We define the extension of data transmitted from an instance-database A into B by the component q_{A_i} as $\|q_{A_i}\| \triangleq \alpha(r_i')$.

Notice that the components $\alpha(v_i)$ and $\alpha(q_i)$ are not the morphisms in **DB** category: only their functional composition is an atomic morphism. Each atomic morphism is a complete morphism, that is, a set of view-mappings. Thus, each view-map $q_{A_i} : A \longrightarrow TA$, which is an atomic morphism, is a complete morphism (the case when $B = TA$, and $\alpha(v_i)$ belongs to the 'exact case'). We denote the set of all complete morphisms by *c-arrow*.

Based on atomic morphisms (sets of view-mappings), which are complete arrows (c-arrows), we obtain that their composition generates tree-structures. These tree-structures can be incomplete (p-arrows) in the way that for a composed arrow

$h = g \circ f : A \to C$, of two atomic arrows $f : A \to B$ and $g : B \to C$, we can have the situations where $\partial_0(f) \subset \partial_0(h)$. The set of relations in $\partial_0(h) - \partial_0(f) \subset \partial_0(g)$ are denominated 'hidden elements'.

Definition 3: The following BNF defines the set Mor_{DB} of all morphisms in DB:

$$
\begin{aligned}
&p - arrow := c - arrow \mid c - arrow \circ c - arrow\\
&\text{(for any two } c - arrows\ f : A \longrightarrow B \text{ and } g : B \longrightarrow C\ )\\
&\text{morphism } := p - arrow \mid c - arrow \circ p - arrow\\
&\text{(for any } p - arrowf : A \longrightarrow B \text{ and } c - arrowg : B \longrightarrow C)
\end{aligned}
$$

whereby the composition of two arrows, f (partial) and g (complete), we obtain the following p-arrow (partial arrow) $h = g \circ f : A \longrightarrow C$

$$h = g \circ f = \bigcup_{q_{B_j} \in\ g\ \&\ \partial_0(q_{B_j}) \bigcap \partial_1(f) \neq \emptyset} \{q_{B_j}\} \quad \circ$$

$$\circ \bigcup_{q_{A_i} \in\ f\ \&\ \partial_1(q_{A_i}) = \{v\}\ \&\ v \in\ \partial_0(q_{B_j})} \{q_{A_i}(tree)\}$$

$$
\begin{aligned}
&= \{q_{B_j} \circ \{q_{A_i}(tree) \mid \partial_1(q_{A_i}) \subseteq \partial_0(q_{B_j})\} \mid q_{B_j} \in\ g\ \&\ \partial_0(q_{B_j}) \bigcap \partial_1(f) \neq \emptyset\}\\
&= \{q_{B_j}(tree) \mid q_{B_j} \in\ g\ \&\ \partial_0(q_{B_j}) \bigcap \partial_1(f) \neq \emptyset\}
\end{aligned}
$$

where $q_{A_i}(tree)$ is the tree of the morphisms f below q_{A_i}.

We define the semantics of mappings by function $B_T : Mor_{DB} \longrightarrow Ob_{DB}$ which, given any mapping morphism $f : A \longrightarrow B$, returns the set of views ('information flux') which are really 'transmitted' from the source to the target object.

1 for atomic morphism,

$$\widetilde{f} = B_T(f) \ \triangleq T\{\|f_i\| \mid f_i \in f\}$$

2 let $g : A \to B$ be a morphism with a flux $\widetilde{g}$ and $f : B \to C$ an atomic morphism with flux $\widetilde{f}$ defined in point 1, then

$$\widetilde{f \circ g} = B_T(f \circ g) \ \triangleq \widetilde{f} \bigcap \widetilde{g}.$$

We introduce an equivalence relation over the morphisms as:

$$f \approx g \quad iff \quad \widetilde{f} = \widetilde{g}.$$

Notice that between any two databases A and B there is at least an 'empty' arrow $\emptyset : A \to B$ such that $\partial_0(\emptyset) = \partial_1(\emptyset) = \widetilde{\emptyset} = \{\perp\} = \perp^0$.

As demonstrated in Majkić (2003a) and Majkić (2008), the database category, **DB**, has the following basic properties:

1 Symmetry: a bijective correspondence between arrows and objects.

2 Duality: **DB** is equal to its dual $\mathbf{DB}^{OP}$, so that each limit is also colimit [for example, product is also coproduct, pullback is also pushout, empty database $\perp^0$ is zero objet, (that is, both initial and terminal object), and so on].

3 It is a 2-category.

Generally, database mappings are not simply programs from values (relations) into computations (views) but an equivalence of computations. Hence each mapping, from any two databases A and B, is symmetric and gives a duality property to the category **DB**. The denotational semantics of database mappings is given by morphisms of the Kleisli category $\mathbf{DB}_T$ which may be 'internalised' in **DB** category as 'computations' (Majkić and Prasad, 2010).

The product $A \times B$ of the databases A and B is equal to their coproduct $A + B$, and the semantics for these product and coproduct operations specify that we can not define a view by using relations of both databases, that is, these two databases have independent DBMS for query evaluation. For example, the creation of an exact copy of a database A in another DB server corresponds to the database $A + A$.

The duality property for products and coproducts are given by the following commutative diagram:

$$\begin{tikzcd}
A \arrow[d, hook, "in_A"'] \arrow[dr, "f"] & & A \\
A+B \arrow[r, "k"] & C \arrow[ur, "f^{OP}"] \arrow[r, "k^{OP}"] \arrow[dr, "g^{OP}"] & A \times B \arrow[u, two heads, "p_A = in_A^{OP}"'] \arrow[d, two heads, "p_B = in_B^{OP}"] \\
B \arrow[u, hook, "in_B"] \arrow[ur, "g"'] & & B
\end{tikzcd}$$

The work presented in Majkić (2009a, 2009b) has considered some relationships of **DB** and standard **Set** category and also introduced the categorial (functors) semantics for two basic database operations, *matching* $\otimes$ and *merging* $\oplus$, such that for any two databases A and B, $A \otimes B = TA \bigcap TB$ and $A \oplus B = T(A \bigcup B)$. That work has also defined the algebraic database lattice and shown that **DB** is concrete, small and locally finitely presentable (lfp) category. Moreover, that work has also:

a shown that DB is also V-category enriched over itself

b developed a metric space and a subobject classifier for this category

c demonstrated that **DB** is a weak monoidal topos.

In this paper we develop a functorial semantics for the database schema mapping system, based on the theory of sketches.

The rest of the paper is organised as follows: Section 2 presents the two basic operations, separation and federation, for database schemas used in database mapping systems and explains why the functorial semantics for database mappings needed a new base category instead of common **Set** category. Section 3 presents a definition of the graph G for a schema database mapping system and the definition of its sketch category $\mathbf{Sch}(G)$. Based on this framework, functorial semantics for database mapping systems with the base category **DB** is presented. Finally, Section 4 concludes this research.

2 Basic database schema operations: separation and federation

For the composition of complex database mapping graphs, it is important to distinguish two basic compositions of two database schemas $\mathcal{A}$ and $\mathcal{B}$ with respect to DBMSs:

- In the composed schema, in order to make it impossible to write a query over the composition with relations of both databases, the two database schemas should be mutually separated by two independent DBMSs: it is a common case when two databases are *separated*, and this symmetric binary separation-composition at schema level are denoted by $\mathcal{A} \dagger \mathcal{B}$, such that $\pi_i(\mathcal{A} \dagger \mathcal{B}) = \pi_i(\mathcal{A}) \biguplus \pi_i(\mathcal{B}), i = 1, 2$.
- In the composed schema, when the two database schemas are *connected* into the same DBMS (without any change of the two original database schemas): in this case, we are able to use the queries over this composed schema with relations *of both* databases for inter database mappings, and this symmetric binary federation-composition at schema level is denoted by $\mathcal{A} \bigoplus \mathcal{B}$, such that $\pi_i(\mathcal{A} \bigoplus \mathcal{B}) = \pi_i(\mathcal{A}) \biguplus \pi_i(\mathcal{B}), i = 1, 2$.

The identity '$=$' for database schemas is naturally defined as follows: for any two $\mathcal{A}, \mathcal{B} \in \mathbb{S}$ ($\mathbb{S}$ is the set of schemas), $\mathcal{A} = \mathcal{B}$ if $\pi_i(\mathcal{A}) = \pi_i(\mathcal{B}), i = 1, 2$. Notice that both symmetric binary operators, $\dagger$ and $\bigoplus$, for database schemas in $\mathbb{S}$ are associative with identity element $\mathcal{A}_\emptyset$ (nullary operator), so that the algebraic structures $((\mathbb{S}, =), \dagger, \mathcal{A}_\emptyset)$ and $((\mathbb{S}, =), \bigoplus, \mathcal{A}_\emptyset)$ are the monoids.

Let us consider the mappings $\mathcal{M} : \mathcal{A} \dagger \mathcal{B} \to \mathcal{C}$ and $\mathcal{M} : \mathcal{A} \bigoplus \mathcal{B} \to \mathcal{C}$. In the first case, in any query mapping $q(\mathbf{x}) \Rightarrow q_C(\mathbf{x}) \in \mathcal{M}$, all relation symbols in the query $q(\mathbf{x})$ must be of database $\mathcal{A}$ or (mutually exclusive) of database $\mathcal{B}$ and this mapping can be represented by the graph:

$$\begin{tikzcd}
\mathcal{A} \arrow[dr, "\mathcal{M}_A"'] & & \mathcal{B} \arrow[dl, "\mathcal{M}_B"] \\
& \mathcal{C} &
\end{tikzcd}$$

where $\mathcal{M}_A \biguplus \mathcal{M}_B = \mathcal{M}$, while in the case of mapping $\mathcal{M} : \mathcal{A} \bigoplus \mathcal{B} \to \mathcal{C}$ such a decomposition is not possible because we can have a query mapping $q(\mathbf{x}) \Rightarrow q_C(\mathbf{x}) \in \mathcal{M}$ with relation symbols from both databases $\mathcal{A}$ and $\mathcal{B}$.

If we introduce the mappings $\mathcal{M}_1 = \{r_{Ai}(\mathbf{x}_i) \Rightarrow r_{Ai}(\mathbf{x}_i) | r_{Ai} \in \mathcal{A}\}$ and $\mathcal{M}_2 = \{r_{Bi}(\mathbf{y}_i) \Rightarrow r_{Bi}(\mathbf{y}_i) | r_{Bi} \in \mathcal{B}\}$ then we obtain the mapping graph,

$$\begin{tikzcd}
\mathcal{A} \arrow[r, "\mathcal{M}_1"] \arrow[dr, "\mathcal{M}_A"'] & \mathcal{A} \dagger \mathcal{B} \arrow[d, "\mathcal{M}"'] & \mathcal{B} \arrow[l, "\mathcal{M}_2"'] \arrow[dl, "\mathcal{M}_B"] \\
& \mathcal{C} &
\end{tikzcd}$$

that can be seen as a cocone diagram for schema database mappings.

Let us consider another dual example, a mapping $\mathcal{M} : \mathcal{C} \rightarrow \mathcal{A} \dagger \mathcal{B}$. In this case, in any query mapping $q_C(\mathbf{x}) \Rightarrow q(\mathbf{x}) \in \mathcal{M}$, all relation symbols in the query $q(\mathbf{x})$ must be of database $\mathcal{A}$ or (mutually exclusive) of database $\mathcal{B}$ and this mapping can be represented by the graph:

$$\begin{tikzcd}
\mathcal{A} & & \mathcal{B} \\
& \mathcal{C} \arrow[ul, "\mathcal{M}_A"] \arrow[ur, "\mathcal{M}_B"'] &
\end{tikzcd}$$

where $\mathcal{M}_A \biguplus \mathcal{M}_B = \mathcal{M}$.

If we again introduce the mappings $\mathcal{M}_1 = \{r_{Ai}(\mathbf{x}_i) \Rightarrow r_{Ai}(\mathbf{x}_i) | r_{Ai} \in \mathcal{A}\}$ and $\mathcal{M}_2 = \{r_{Bi}(\mathbf{y}_i) \Rightarrow r_{Bi}(\mathbf{y}_i) | r_{Bi} \in \mathcal{B}\}$ then we obtain dual mapping graph,

$$\begin{tikzcd}
\mathcal{A} & \mathcal{A} \dagger \mathcal{B} \arrow[l, "\mathcal{M}_1"'] \arrow[r, "\mathcal{M}_2"] & \mathcal{B} \\
& \mathcal{C} \arrow[ul, "\mathcal{M}_A"] \arrow[u, "\mathcal{M}"] \arrow[ur, "\mathcal{M}_B"'] &
\end{tikzcd}$$

that can be seen as a cone diagram for schema database mappings.

Based on these two simple examples, generally, the schema database mappings can be expressed by using the small *sketches*. A detailed presentation of sketches for the database mappings and their functorial semantics are provided in Section 3.

Sketches are developed by Ehresmann's school, especially by Ehresmann (1966), Lair (1996) and Barr and Wells (1988). Sketch is a category together with a distinguished class of cones and cocones. A *model* of the sketch is a set-valued functor turning all distinguished cones into limit cones, and all distinguished cocones into colimit cocones, in the category **Set** of sets.

There is an elementary and basic connection between sketches and logic (Makkai and Pare, 1989). Given any sketch, we can consider the underlying graph of the sketch as a (many-sorted) language, and we can write down axioms in the $\mathcal{L}_{\infty,\infty}$-logic (the infinitary FOL with finite quantifiers) over this language and hence the models of the axioms become exactly the models of the sketch.

The category of models of a given sketch has models as objects and the arrows that represent all natural transformations between the models as functors. A category is sketchable (esquissable) or accessible iff it is equivalent to the category of set-valued models of a small sketch.

Recall that a graph G consists of a set of vertices denoted by G_0 and a set of arrows denoted by G_1 together with the operators $dom, cod : G_1 \to G_0$ which assigns the source and target to each arrow. (Co)cones and diagrams are defined for graphs exactly in the same way as they are for categories, but commutative co(cones) and diagrams, of course, make no sense for graphs.

We define a sketch as a 4-tuple (G, u, D, C) where G is a graph, $u : G_0 \to G_1$ is a function which takes each vertex (node) $\mathcal{A}$ in G_0 to an arrow from $\mathcal{A}$ to $\mathcal{A}$, D is a class of diagrams in G and C is a class of (co)cones in G. Each (c)cone in G goes (to)from some vertex (from)to some diagram; that diagram need not be in D and, in fact, it is necessary to allow diagrams which are not in D as bases of (co)cones.

Notice that, differently from the work dedicated to categorical semantics of entity-relationship internal relational database models, where nodes of sketches are single relations, here at higher level of abstraction, the nodes are whole databases. Consequently, in such a framework, we do not use commutative database mapping systems, so that D is an empty set. In fact, in a database mapping system, the (co)cone diagrams mentioned above will never be used in practical representations of database mapping systems. Instead of that, it will be alternatively used only for its selfconsistent parts, as a first diagram above, or, equivalently, as a single arrow $\mathcal{M} : \mathcal{A} \dagger \mathcal{B} \to \mathcal{C}$. However, for the introduced schema composition operator $\dagger$, the above cone and cocone diagrams have to be presented in C for our sketches.

Consequently we obtain the following fundamental lemma for the categorial modelling of database mappings:

Lemma 1: The **Set** can not be used as the base category for the models of database-mapping sketches.

Proof: Let **E** be a sketch for a given database sketch (G, u, D, C), where C is a set of (co)cones of the two diagrams introduced for the database schema composition operator $\dagger$, and a model of this sketch be a functor $F : \mathbf{E} \to \mathbb{B}$, where $\mathbb{B}$ is a base category. Then all cones in C have to be functorially translated into limit commutative diagrams in $\mathbb{B}$, and all cocones in C have to be functorially translated into limit commutative diagrams in $\mathbb{B}$: i.e., the cocone in the figure above has to be translated into coproduct diagram, and cone translated into product diagram in $\mathbb{B}$.

Consequently, the object $F(\mathcal{A} \dagger \mathcal{B})$ has to be both the product $A \times B$ and coproduct $A + B$, where $A = F(\mathcal{A})$ and $B = F(\mathcal{B})$ are two objects in $\mathbb{B}$; $\times$ and $+$ are the product and coproduct operators in $\mathbb{B}$, but this can not be done in **Set**. In fact, the product $A \times B$ in **Set** is the Cartesian product of these two sets A and B while the coproduct $A + B$ is the disjoint union. Hence, $A \times B \simeq A + B$ is not an isomorphism in **Set**. □

Remark: The fundamental consequence of this lemma is that we need to define a new base category for the categorial semantics of database mappings. In fact, we defined this new base category $\mathbb{B}$, denoted by **DB** (DataBase) category, and we have shown that it satisfies the duality property, where the product and coproduct diagrams are dual diagrams and hence for any two objects (instance databases) in **DB**, the objects $A \times B$ and $A + B$ are equal (up to isomorphism).

2.1 Data separation: (co)product operator in DB

Separation-composition of objects are coproducts (and products) in **DB** category:

Definition 4: The disjoint union of any two instance-databases (objects) A and B, denoted by $A+B$, corresponds to two mutually isolated databases, where two database management systems are completely disjoint and hence it is impossible to compute the queries with the relations from the both databases.

The disjoint property for mappings is represented by the facts that

$$\partial_0(f+g) \triangleq \partial_0(f)+\partial_0(g) \text{ and } \partial_1(f+g) \triangleq \partial_1(f)+\partial_1(g).$$

Thus, for any database A, the *replication* of this database (over different DB servers) can be denoted by the coproduct object $A+A$ in this category **DB**.

Proposition 1: For any two databases (objects) A and B, $T(A+B)=TA+TB$. Consequently $A+A$ is not isomorphic to A.

Proof: $T(A+B)=TA+TB$ is true because of the fact that we are able to define views only over relations in A or, alternatively, over relations in B. Analogously $\widetilde{f+g}=\widetilde{f}+\widetilde{g}$, which is a closed object, that is,

$$T(\widetilde{f+g})=T(\widetilde{f}+\widetilde{g})=T\widetilde{f}+T\widetilde{g}=\widetilde{f}+\widetilde{g}=\widetilde{f+g}.$$

From $T(A+A)=TA+TA\neq TA$ we obtain that $A+A$ is not isomorphic to A. $\square$

Notice that for coproducts $C+\perp^0 = \perp^0+C \simeq C$ is true, and for any arrow f in **DB**, $f+\perp^1 \approx \perp^1+f \approx f$, where $\perp^1$ is a banal empty morphism between objects and hence $\partial_0(\perp^1)=\partial_1(\perp^1)=\perp^0$, with $\widetilde{\perp^1}=\perp^0$.

We are now ready to introduce the duality property between coproducts and products in this **DB** category:

Proposition 2: There exists an idempotent coproduct bifunctor $+:\mathbf{DB}\times\mathbf{DB}\longrightarrow\mathbf{DB}$ which is a disjoint union operator for objects and arrows in **DB**.

The category **DB** is cocartesian with initial (zero) object $\perp^0$ and for every pair of objects A,B it has a categorial coproduct $A+B$ with monomorphisms (injections) $in_A : A\hookrightarrow A+B$ and $in_B : B\hookrightarrow A+B$.

Based on duality property, **DB** is also Cartesian category with a zero object $\perp^0$. For each pair of objects A,B there exists a categorial product $A\times B$ with epimorphisms (projections) $p_A=in_A^{OP} : A\times A\twoheadrightarrow A$ and $p_B=in_B^{OP} : B\times B\twoheadrightarrow B$, where the product bifunctor is equal to the coproduct bifunctor, i.e., $\times \equiv +$.

Proof:

1 For any identity arrow (id_A, id_B) in $\mathbf{DB}\times\mathbf{DB}$, where id_A and id_b are the identity arrows of A and B respectively,

$$\widetilde{id_A+id_B}=\widetilde{id_A}+\widetilde{id_B}=TA+TB=T(A+B)=\widetilde{id_{A+B}} \text{ is true.}$$

Thus, $+^1(id_A, id_B)=id_A+id_B=id_{A+B}$ is an identity arrow of the object $A+B$.

2 For any

$$k : A \longrightarrow A_1,\ k_1 : A_1 \longrightarrow A_2,\ l : B \longrightarrow B_1,\ l_1 : B_1 \longrightarrow B_2,$$
$$\widetilde{+^1(k_1,l_1) \circ +^1(k,l)} = \widetilde{+^1(k_1,l_1)} \bigcap \widetilde{+^1(k,l)} = \widetilde{k_1 \circ k} + \widetilde{l_1 \circ l}$$
$$= \widetilde{+^1(k_1 \circ k,\ l_1 \circ l)} = \widetilde{+^1((k_1,k) \circ (l_1,l))},$$

thus $+^1(k_1,l_1) \circ +^1(k,l) = +^1((k_1,k) \circ (l_1,l))$.

3 Let us demonstrate the coproduct property of this bifunctor: for any two arrows $f : A \longrightarrow C$, $g : B \longrightarrow C$, there exists a unique arrow $k : A + B \longrightarrow C$, such that $f = k \circ in_A$, $g = k \circ in_B$, where $in_A : A \hookrightarrow A + B$, $in_B : B \hookrightarrow A + B$ are the injection (point to point) monomorphisms ($\widetilde{in_A} = TA$, $\widetilde{in_B} = TB$).

It is easy to verify that for any two arrows $f : A \longrightarrow C$, $g : B \longrightarrow C$, there exists exactly one arrow $k = e_C \circ (f + g) : A + B \longrightarrow C$, where $e_C : C + C \twoheadrightarrow C$ is an epimorphism (with $\widetilde{e_C} = TC$), such that $\widetilde{k} = \widetilde{f} + \widetilde{g}$. □

2.2 *Data federation operator in DB*

The opposite operation to (co)product (a DBMS's separation) is the DBMS's *data federation* of two database instances A and B. In this way we are able to compute the queries with the relations of *both* databases. In fact, data federation technology is just used for such an integration of two previously separated databases.

Consequently, given any two databases (objects in **DB**) A and B, the federation of them (under the common DBMS) corresponds to *union* of them under the same DBMS, thus, equals to database $A \bigcup B$.

3 Categorial semantics of database schema mappings

It is natural for the database schema $\mathcal{A} = (S_A, \Sigma_A)$, where S_A is a set of n-ary relation symbols and Σ_A are the database integrity constraints, to take Σ_A to be a *tuple-generating dependency* (*tgd*) and *equality-generating dependency* (*egd*). We denote the empty database schema with empty set of relation symbols by $\mathcal{A}_\varnothing$, where $\Sigma_{\mathcal{A}_\varnothing}$ is the empty set of integrity constraints.

A tgd specifies that if some tuples satisfying certain equalities exist in the relation then some other tuples (possibly with some unknown values) must also exist in the relation.

An egd specifies that if some tuples satisfying certain equalities exist in the relation then some values in these tuples must be equal. Functional dependencies are egds of a special form, for example primary-key integrity constraints. These two classes of dependencies together comprise the *embedded implication dependencies* (EID) (Fagin, 1982) which seem to include all naturally-occuring constraints on relational databases (the bold symbols $\mathbf{x}, \mathbf{y}, \ldots$ denote a non-empty list of variables):

1 a *tuple-generating dependency* (*tgd*) of the FOL form

$$\forall \mathbf{x}\ (\exists \mathbf{y}\ \phi_A(\mathbf{x},\mathbf{y})\ \Rightarrow\ \exists \mathbf{z}\ (\psi_{\mathbf{A}}(\mathbf{x},\mathbf{z}))$$

where the formulae $\phi_A(\mathbf{x},\mathbf{y})$ and $\psi_A(\mathbf{x},\mathbf{z})$ are conjunctions of atomic formulas over $\mathcal{A}$ (for integrity constraints over database schemas, we consider only class of weakly-full tgd for which query answering is decidable, i.e., when the right-hand side has no existentially quantified variables and if each $y_i \in \mathbf{y}$ appears at most once in the left-hand side).

2 an *equality-generating dependency* (egd):

$$\forall \mathbf{x}\ (\phi_A(\mathbf{x})\ \Rightarrow\ (\mathbf{x_1} = \mathbf{x_2}))$$

where a formula $\phi_A(\mathbf{x})$ is a conjunction of atomic formulas over $\mathcal{A}$; $\mathbf{x}_1$ and $\mathbf{x}_2$ are among the variables in $\mathbf{x}$.

Notice that any schema database *mapping* from a schema $\mathcal{A}$ into a schema $\mathcal{B}$ is represented by the general tgd $\forall \mathbf{x}\ (\exists \mathbf{y}\ \phi_A(\mathbf{x},\mathbf{y})\ \Rightarrow\ \exists \mathbf{z}\ \psi_B(\mathbf{x},\mathbf{z}))$, that is by the *view mapping* $q_A(\mathbf{x}) \Rightarrow\ q_B(\mathbf{x})$, as used in Definition 7 provided later on, where $q_A(\mathbf{x})$ (equivalent to $\exists \mathbf{y}\ \phi_A(\mathbf{x},\mathbf{y})$ is a query over the schema $\mathcal{A}$, and $q_B(\mathbf{x})$ (equivalent to $\exists \mathbf{z}\ \psi_B(\mathbf{x},\mathbf{y})$) is a query over the schema $\mathcal{B}$.

Next, we explain how the logical *model* theory for database schemas and their mappings based on views can be translated into the category theory by using the **DB** category defined in the previous section. The integrity constraints for databases are expressed by the FOL logical sentences (i.e., the FOL formulae without free variables). Such sentences are expressed in the schema database level by the mappings from the database schema $\mathcal{A}$ into the empty database schema $\mathcal{A}_\varnothing$. We define their denotation in the **DB** category as follows:

Definition 5: For any sentence $\varphi : \mathcal{A} \rightarrow \mathcal{A}_\varnothing$ (a logical formula without variables, in Definition 1) over a database schema $\mathcal{A}$ and a given interpretation α such that φ is satisfied by it, then there exists the unique morphism from $\mathcal{A}$ into terminal object $\perp^0$ in **DB** category, $f : A \rightarrow \perp^0$, where $f = \alpha^*(\varphi)$ and $A = \alpha^*(\mathcal{A})$ (for $A \simeq \perp^0$ as well). Otherwise, when $A = \alpha^*(\mathcal{A})$ is not isomorphic to $\perp^0$ and if φ is not satisfied by α then $\alpha^*(\varphi)$ is mapped into the identity arrow $id_{\perp^0} : \perp^0 \rightarrow \perp^0$.

Notice that unlike view-mappings for queries (formulae with free variables) given in Definition 2, the integrity constraints in a **DB** category have the empty database (zero object, i.e., terminal and initial) as the codomain, and the information flux equals to $\perp^0 = \{\perp\}$. It is consistent with the definition of morphisms in **DB** category because the sentences do not transfer any data from source to target database and hence their information flux has to be empty. In the case of ordinary query mappings, the minimal information flux is $\{\perp\} = \perp^0$ as well. In **DB** category, for any unique morphism from initial object $\perp^0$ (empty database) to another object (database) A, $f : \perp^0 \rightarrow A$, the information flux of these morphisms is also equal to $\perp^0$.

Based on these semantics for logical formulae without free variables (integrity constraints and Yes/No queries), we are able to define the categorial interpretations for database schema mappings, as follows.

3.1 Categorial semantics of database schemas

As we explained in Section 2, in order to define the database mapping systems, we use two fundamental operators for the database schemas, data federation $\bigoplus$ and data separation $\dagger$, with the two correspondent monoids, $((\mathbb{S},=),\dagger,\mathcal{A}_\emptyset)$ and $((\mathbb{S},=),\bigoplus,\mathcal{A}_\emptyset)$, and the distribution law: $\mathcal{A}\bigoplus(\mathcal{B}\dagger\mathcal{C}) = (\mathcal{A}\bigoplus\mathcal{B})\dagger(\mathcal{A}\bigoplus\mathcal{C})$.

Consequently, each vertex in the graph G of a database mapping system is a term of the combined algebra of these two monoids, $\mathbb{S}_{Alg} = ((\mathbb{S},=),\bigoplus,\dagger,\mathcal{A}_\emptyset)$.

In the rest of the paper, we denote the database schema for any well formed *term* (i.e., an algebraic expression) of this algebra for schemas $\mathbb{S}_{Alg}$. We also denote by $\mathcal{A}\in\mathbb{S}_{Alg}$ a database schema that can be either an *atomic* schema or composed schema by a finite number of atomic schemas and two algebraic operators $\bigoplus$ and data separation $\dagger$ of the algebra $\mathbb{S}_{Alg}$.

Consequently for *each* schema $\mathcal{A}\in\mathbb{S}_{Alg}$, we have $\mathcal{A}=(S_A,\Sigma_A)$, where $S_A = \biguplus\{S_{B_i} \mid \mathcal{B}_i$ is an atomic schema in the schema expression $\mathcal{A}\}$ and $\Sigma_A = \biguplus\{\Sigma_{B_i} \mid \mathcal{B}_i$ is an atomic schema in the schema expression $\mathcal{A}\}$.

For each atomic schema database and an interpretation α, $A=\alpha^*(\mathcal{A})$ is an instance-database of this schema and thus it is an object in **DB** category. The interpretation of the composite schemas (i.e., the non-atomic terms of the algebra $\mathbb{S}_{Alg}$) in **DB** category is given by the following proposition:

Proposition 3: Let α be a given interpretation then there exists the following homomorphism from the schema database level into the instance database level:

$$\alpha^* : ((\mathbb{S},=),\bigoplus,\dagger,\mathcal{A}_\emptyset) \to ((Ob_{DB},\simeq),\biguplus,+,\bot^0).$$

Proof: The interpretation of a given schema $\mathcal{A}$ is an instance $A=\alpha^*(\mathcal{A})$ of this database, that is an object in **DB**; for every interpretation $\alpha^*(\mathcal{A}_\emptyset)=\bot^0$.

From the monoidal property we have the equation $\mathcal{A}\bigoplus\mathcal{A}_\emptyset=\mathcal{A}$ in the algebra $\mathbb{S}_{Alg}$. By the above homomorphism, we have that $\alpha^*(=) = \simeq$ and $\alpha^*(\bigoplus)=\biguplus$, so that $\alpha^*(\mathcal{A}\bigoplus\mathcal{A}_\emptyset)=\alpha^*(\mathcal{A})\biguplus\alpha^*(\mathcal{A}_\emptyset)=A\biguplus\bot^0\simeq A$. From the monoidal property we have the equation $\mathcal{A}\dagger\mathcal{A}_\emptyset=\mathcal{A}$ in the algebra $\mathbb{S}_{Alg}$. By the homomorphism above we have $\alpha^*(\dagger)=+$, so that $\alpha^*(\mathcal{A}\dagger\mathcal{A}_\emptyset)=\alpha^*(\mathcal{A})+\alpha^*(\mathcal{A}_\emptyset)=A+\bot^0\simeq A$ is an isomorphism in **DB**. □

Let $\mathcal{A}=(S_A,\Sigma_A)$ be the database schema, where S_A is a set of relation symbols with a given list of attributes and $\Sigma_A=\Sigma_A^{tgd}\bigcup\Sigma_A^{egd}=\pi_2(\mathcal{A})$ are the database integrity constraints (set of EIDs) which can be an empty set as well. We can represent the integrity constraints by a sketch schema mapping $\phi_A:\mathcal{A}\longrightarrow\mathcal{A}_\varnothing$ (ϕ_A denotes the sentence obtained by conjunction of all formulae in Σ_A), where $\mathcal{A}_\varnothing$ is the empty schema, such that for any interpretation α it holds that $\alpha(\mathcal{A}_\varnothing)=\bot^0$.

Proposition 4: If there exists a model A for a database schema $\mathcal{A}=(S_A,\Sigma_A)$ which satisfies all integrity constraints $\Sigma_A=\Sigma_A^{tgd}\bigcup\Sigma_A^{egd}$ (ϕ_A denotes the sentence obtained by conjunction of all formulae in Σ_A) then there exists the following interpretation R-algebra α and its extension, the functor $\alpha^*:\mathbf{Sch}(G)\longrightarrow\mathbf{DB}$, where $\mathbf{Sch}(G)$ is the sketch category derived from the graph G with the arrow $\Sigma_A:\mathcal{A}\longrightarrow\mathcal{A}_\varnothing$ (i.e., $\mathbf{Sch}(G)$

is composed of the objects $\mathcal{A}$, $\mathcal{A}_\varnothing$, the arrow $\phi_A : \mathcal{A} \longrightarrow \mathcal{A}_\varnothing$ and the identity arrows $id_\mathcal{A} : \mathcal{A} \longrightarrow \mathcal{A}$ and $id_{\mathcal{A}_\varnothing} : \mathcal{A}_\varnothing \longrightarrow \mathcal{A}_\varnothing$), such that:

1 $\alpha^*(\mathcal{A}) \triangleq A$, where A is possibly an empty database instance (with all empty relations) as well

2 $\alpha^*(id_\mathcal{A}) \triangleq id_A : A \longrightarrow A$

3 $\alpha^*(id_{\mathcal{A}_\varnothing}) \triangleq id_{\perp^0} : \perp^0 \longrightarrow \perp^0$

4 $\alpha^*(\phi_A) \triangleq (f_{tgd} \bigcup f_{egd}) : A \longrightarrow \perp^0$.

Proof: From Definition 5 and Point 4 of this proposition, each integrity constraint $q_i \in \Sigma_A$ of the database schema $\mathcal{A}$ is satisfied by the interpretation α (because the conjunction of all integrity constraints, denoted by ϕ_A, is satisfied w.r.t the Definition 5): if Σ_A is empty then it is always satisfied as usual. Thus α is a model of a database schema $\mathcal{A}$ and the instance of this model is the non-empty database $A = \alpha^*(\mathcal{A})$, that is an object in the **DB** category. □

Notice that any empty database A (that is, all its relations are empty) is isomorphic to the database $\perp^0$ with only one empty relation $\perp$ (i.e., $\perp^0 = \{\perp\}$). It is easy to show because any arrow for this empty database $f : A \rightarrow A$ has the information flux $\widetilde{f} = \perp^0$ and hence $f = id_A$ is the unique identity arrow for this empty database. However, the unique arrows $g : A \rightarrow \perp^0$ and $h : \perp^0 \rightarrow A$ have the same information fluxes, i.e., $\widetilde{g} = \widetilde{h} = \perp^0$ and hence $g \circ h = id_{\perp^0}$ and $h \circ g = id_A$, and, consequently, $A \simeq \perp^0$.

Consequently, the remark in point 1 of this proposition specifies that if $A = \alpha^*(\mathcal{A}) \simeq \perp^0$ is an empty database then:

1 α^* is a model of a schema $\mathcal{A}$

2 integrity constraint for point 4 in this proposition corresponds to the satisfaction of this integrity constraint w.r.t. the Definition 5.

3.2 Categorial semantics of database mappings

First, we formally define a schema database mapping *graph G*, as follows:

Definition 6: A schema database mapping *graph G* is composed of an atomic arrow $\Sigma_A : \mathcal{A} \rightarrow \mathcal{A}_\emptyset$ for each database schema $\mathcal{A}$, and a number of (a view-based) atomic schema database mappings $\mathcal{M} : \mathcal{A} \rightarrow \mathcal{B}$ between two given schemas A and $\mathcal{B}$.

We use the following basic binary operators for these database mapping graphs:

- Given two mappings $\mathcal{M}_1 : \mathcal{A} \rightarrow \mathcal{B}$ and $\mathcal{M}_2 : \mathcal{B} \rightarrow \mathcal{C}$, we denote their sequential composition in this graph G by $\mathcal{M}_2; \mathcal{M}_1$, where ; is a binary associative but a non-commutative operator.
- Given two mappings $\mathcal{M}_1 : \mathcal{A} \rightarrow \mathcal{B}$ and $\mathcal{M}_2 : \mathcal{A} \rightarrow \mathcal{C}$, we denote their branching in this graph G by $\mathcal{M}_2 \biguplus \mathcal{M}_1 : \mathcal{A} \rightarrow B \dagger C$, where $\biguplus$ is a binary associative and commutative operator of disjoint union.

It is easy to verify that a graph G can be extended into a sketch category $\mathbf{Sch}(G)$.

The semantics of a view-based mapping $\mathcal{M} = \{q_{Ai}(\mathbf{x}_i) \Rightarrow q_{Bi}(\mathbf{x}_i) | 1 \leq i \leq n\}$ from a relational database schema $\mathcal{A}$ into a database schema $\mathcal{B}$ are constraints on the pairs of interpretations, of $\mathcal{A}$ and $\mathcal{B}$, and therefore specify which pairs of interpretations can co-exist (i.e., also satisfy the mapping between schemas), given the mapping (see Madhavan et al., 2002 also). The formalisation of the embedding $\gamma : G \rightarrow \mathbf{Sch}(G)$ of a graph G into the sketch $\mathbf{Sch}(G)$ can be given by iteration of the following rules:

Definition 7: We consider the view-based mappings between schemas defined in the SQL language of $SPJRU$ algebra. The arrows in the sketch $\mathbf{Sch}(G)$, for any arrow $\mathcal{M} : \mathcal{A} \rightarrow \mathcal{B}$ in a given graph G in Definition 6, where $\mathcal{M} = \{q_{Ai}(\mathbf{x}_i) \Rightarrow q_{Bi}(\mathbf{x}_i) | 1 \leq i \leq n\}$ and $q_{Ai}(\mathbf{x}_i), q_{Bi}(\mathbf{x}_i)$ are open FOL formulae over $\mathcal{A}$, are defined as follows:

1 for each $q_{Ai}(\mathbf{x}_i) \Rightarrow q_{Bi}(\mathbf{x}_i) \in \mathcal{M}$ such that that q_{Bi} is not a relation symbol of a database schema $\mathcal{B}$, we introduce a new relation $r_i(\mathbf{x}_i)$ in $\gamma(\mathcal{B})$ (we use the same symbol for this γ-enlarged database schema by these new relations). Then we introduce the single mapping arrow $f_{\mathcal{M}} : \mathcal{A} \rightarrow \mathcal{B}$ in $\mathbf{Sch}(G)$, where

$$f_{\mathcal{M}} = \gamma(\mathcal{M}) \triangleq \bigcup_{1 \leq i \leq n} \{v_i \cdot q_i \ : \mathcal{A} \longrightarrow \mathcal{B} \mid \partial_0(q_i) = R_i,$$
$$\partial_1(q_i) = \partial_0(v_i),\ \partial_1(v_i) = \{r_i\}\}$$

and q_i and v_i are abstract 'operations' (operads) introduced in Definition 2, such that for a given model α of this database schema mapping, $\alpha(q_i)$ is a query computation of a query $q_{Ai}(\mathbf{x}_i)$. The set R_i is the set of relation symbols in $\mathcal{A}$ used in the formula $q_{Ai}(\mathbf{x}_i)$.

2 for each $q_{Ai}(\mathbf{x}_i) \Rightarrow q_{Bi}(\mathbf{x}_i) \in \mathcal{M}$ such that q_{Bi} is not a relation symbol of a database schema $\mathcal{B}$ (then $q_{Ai}(\mathbf{x}_i) \Rightarrow q_{Bi}(\mathbf{x}_i)$ (logical implication between queries) means that each tuple of the view obtained by the query $q_{Ai}(\mathbf{x}_i)$ is also a tuple of the view obtained by the query $q_{Bi}(\mathbf{x}_i)$), we do as follows:

In this sketch $\mathbf{Sch}(G)$, we introduce a new helper database schema $\mathcal{C}_i$ with a single relation $c_i(\mathbf{x}_i, y)$ and two new schema mappings:

$f_{AC_i} = w_i \cdot q_i : \mathcal{A} \rightarrow \mathcal{C}_i$ (with $\partial_0(w_i) = \partial_1(q_i)$ and $\partial_1(w_i) = \{c_i\}$) and $f_{BC_i} = w'_i \cdot q'_i : \mathcal{B} \rightarrow \mathcal{C}_i$ (with $\partial_0(q'_i) = R'_i$, $\partial_1(q'_i) = \partial_0(w'_i)$ and $\partial_1(w'_i) = \{c_i\}$, where R'_i is the set of all relation symbols in $\mathcal{B}$ used in the formula $q_{Bi}(\mathbf{x}_i)$) such that f_{AC_i} corresponds to $\{q_{Ai}(\mathbf{x}_i) \Rightarrow c_i(\mathbf{x}_i, \natural_A)\}$ and f_{BC_i} corresponds to $\{q_{Bi}(\mathbf{x}_i) \Rightarrow c_i(\mathbf{x}_i, \natural_B)\}$, where $\natural_A$ and $\natural_B$ are two new values not present in the domain of the databases.

Consequently, in G, we also introduce the integrity constraint arrow $\varphi_i : \mathcal{C}_i \rightarrow \mathcal{A}_\varnothing$ for this new schema $\mathcal{C}_i$, where the sentence φ_i is equal to the tgd

$$\forall \mathbf{x}_i\ (\exists y(c_i(\mathbf{x}_i, y) \wedge y = \natural_A)\ \Longrightarrow\ \exists z(c_i(\mathbf{x}_i, z) \wedge z = \natural_B)).$$

It is easy to verify that there is at most *one* arrow between any two nodes in the obtained sketch **Sch**(G). There is a fundamental functorial *interpretation* connection between schema mappings and their models in the instance level category **DB**. It is based on the Lawvere (1963) categorial theories (Barr and Wells, 1985) in which he introduced a way of describing algebraic structures using categories for theories, functors (into base category **Set**, which we substitute by more adequate category **DB**), and natural transformations for morphisms between models.

For example, Lawvere's seminal observation is that the theory of groups is a category with group object, that a group in **Set** is a product preserving functor, and that a morphism of groups is a natural transformation of functors. This observation was successively extended to define the categorial semantics for different algebraic and logic theories.

This work is based on the theory of *sketches*, which are fundamentally small categories obtained from graphs enriched with concepts such as (co)cones mapped by functors in (co)limits of the base category **Set**. It was demonstrated that, for every sentence in basic logic, there is a sketch with the same category of models and vice versa (Makkai and Pare, 1989). Accordingly, sketches are called graph-based logic and provide very clear and intuitive specification of computational data and activities.

For any small sketch **E**, the category of models $Mod(\mathbf{E})$ is an accessible category by Lair's theorem and reflexive subcategory of $\mathbf{Set}^{\mathbf{E}}$ by Ehresmann-Kennison theorem. A generalisation to base categories other than **Set** was proved by Freyd and Kelly (1972). In rest of the paper, we substitute the base category **Set** by this new database category **DB**.

For instance, for the separation-composition mapping cocone diagram (graph G), given in the introduction, its translation in a *sketch* (a category **Sch**(G)) is in the left-hand side commutative diagram presented below (notice that the mapping arrow $\mathcal{M}$ in a graph G is replaced by the morphism $f_{\mathcal{M}}$ in this sketch, while the nodes (objects) are changed eventually by introducing another auxiliary relation symbols as explained in Definition 7), and the functorial translation of this sketch into **DB** category has to be coproduct diagram in **DB** as follows:

$$\begin{tikzcd}
\gamma(\mathcal{A}) \arrow[r, "f_{\mathcal{M}_1}"] \arrow[dr, "f_{\mathcal{M}_A}"'] & \gamma(\mathcal{A}) \dagger \gamma(\mathcal{B}) \arrow[d, "f_{\mathcal{M}}"] & \gamma(\mathcal{B}) \arrow[l, "f_{\mathcal{M}_2}"'] \arrow[dl, "f_{\mathcal{M}_\mathcal{B}}"] & \Rrightarrow & A \arrow[r, hook, "In_A"] \arrow[dr, "f"'] & A+B \arrow[d, "k"] & B \arrow[l, hook', "In_B"'] \arrow[dl, "g"] \\
& \gamma(\mathcal{C}) & & & & C &
\end{tikzcd}$$

As we explained in the introduction, in database mapping systems, expressed by a graph G, we never use 'commutative diagrams' as left diagram above (but only an arrow $f_{\mathcal{M}} : \gamma(\mathcal{A}) \dagger \gamma(\mathcal{B}) \to \gamma(\mathcal{C})$, or, more frequently, two simple arrows $f_{\mathcal{M}_A} = \gamma(\mathcal{M}_A) : \gamma(\mathcal{A}) \to \gamma(\mathcal{C})$ and $f_{\mathcal{M}_B} = \gamma(\mathcal{M}_B) : \gamma(\mathcal{B}) \to \gamma(\mathcal{C})$), our sketch $\mathbf{E} = \mathbf{Sch}(G)$ is a simple small category, i.e., 4-tuple (G, u, D, C) where D and C are empty sets. Consequently, these database-mapping sketches are more simple than the sketches used for definition of Entity-Relationship models of single relational databases.

Proposition 5: Let $\mathbf{Sch}(G)$ be a schema sketch category generated from a schema mapping graph G that is obtained by applying the method in Definition 7 for each mapping between two database schemas in a given database mapping system with $n \geq 2$ database schemas. Let an interpretation R-algebra α satisfies the following property: for any database schema $\mathcal{A}$ (object in $\mathbf{Sch}(G)$), α satisfies the Proposition 4 so that $A \triangleq \alpha^*(\mathcal{A}) \in Ob_{DB}$ is a model of the database schema $\mathcal{A}$ and for each schema mapping arrow $f_{Sch} : \mathcal{A} \longrightarrow \mathcal{B}$, (where $\mathcal{B}$ is not empty schema) the atomic morphism in **DB** category $\alpha^*(f_{Sch}) : \alpha^*(\mathcal{A}) \to \alpha^*(\mathcal{B})$ is determined by banal *set-inclusion case* of Definition 2.

Now there is the functor (categorial model)

$$\alpha^* : \mathbf{Sch}(G) \longrightarrow \mathbf{DB} \ .$$

The set of categorial models of the database schema mapping graph G is equal to the homset $hom(\mathbf{Sch}(G), \mathbf{DB})$ of all functors from these two categories in the category **Cat**, i.e., equal to the set of all objects in the category of functors $\mathbf{DB}^{\mathbf{Sch}(G)}$ as well.

For a given model (functor) $\alpha^* \in \mathbf{DB}^{\mathbf{Sch}(G)}$, its image in **DB** is called a `DB-mapping' \verbsystem' and is denoted by $\mathcal{M}_S$.

Proof: This is easy to verify based on general theory for sketches (Barr and Wells, 1985). Each arrow in a sketch (obtained from a schema mapping graph G) may be converted into a tree syntax structure of some morphism in **DB** (labelled tree without any interpretation). The functor α^* is only a simple extension of the interpretation R-algebra function α for a lists of symbols. The functorial property for the identity mappings follows from Proposition 4. For any two atomic mappings, $f_{Sch} : \mathcal{A} \longrightarrow \mathcal{B}$ and $g_{Sch} : \mathcal{B} \longrightarrow \mathcal{C}$, and their atomic morphisms in **DB**, $f = \alpha^*(f_{Sch})$ and $g = \alpha^*(g_{Sch})$, we have that $\alpha^*(g_{Sch} \circ f_{Sch}) = g \circ f$ as defined in Definition 3. It remains only to verify that for each auxiliary database schema $\mathcal{C}_i$ and integrity constraint $\varphi_i : \mathcal{C}_i \to \mathcal{A}_\emptyset$ (in Definition 7), the operator α^* satisfies the functorial property such that this schema arrow is mapped into the arrow $\alpha^*(\varphi_i) : C_i \to \perp^0$, where $C_i = \alpha^*(\mathcal{C}_i)$. In fact it is true because if α is a model of this database mapping system represented by the graph G then this integrity constraint φ_i is satisfied and, based on Definition 5, the functorial property is satisfied.

Notice that this proposition is also true for the special cases when $C_i = \alpha^*(\mathcal{C}_i) \simeq \perp^0$. This special case occurs when, for a view mapping $q_A(\mathbf{x}) \Rightarrow q_B(\mathbf{x})$, both $\|q_A(\mathbf{x})\|$ (resulting view of the query $q_A(\mathbf{x})$ over the database $A = \alpha^*(\mathcal{A})$) and $\|q_B(\mathbf{x})\|$ (resulting view of the query $q_B(\mathbf{x})$ over the database $B = \alpha^*(\mathcal{B})$) are empty relations and consequently $\alpha(c_i(\mathbf{x}, y))$ is an empty relation in the database instance $C_i = \alpha^*(\mathcal{C}_i) = \{\alpha(c_i(\mathbf{x}, y))\}$ and hence $C_i \simeq \perp^0$. Thus an integrity constraint $\varphi_i : \mathcal{C}_i \to \mathcal{A}_\emptyset$ (an auxiliary arrow in $\mathbf{Sch}(G)$ obtained from some mapping between two database schemas in G) can be unsatisfied only if $C_i = \alpha^*(\mathcal{C}_i)$ is not isomorphic to $\perp^0$. In order to prove that the set of functors in $\mathbf{DB}^{\mathbf{Sch}(G)}$ is exactly the set of all models of the database mapping system expressed by the graph G, it is now enough to prove that any interpretation α that *is not* a model of G then that interpretation can not be a functor from $\mathbf{Sch}(G)$ into **DB**. In order that a given α is not a model of G, the α must satisfy the following cases:

1 For some database schema $\mathcal{A}$ in G, $\alpha^*(\mathcal{A})$ is not a model of this database. It means that the conjunction of all integrity constraints $\Sigma_A : \mathcal{A} \to \mathcal{A}_\emptyset$ of $\mathcal{A}$ is not satisfied by α (thus when $\alpha^*(\mathcal{A})$ not isomorphic to $\perp^0$, as specified by Definition 5) so that, from Definition 5, $\alpha^*(\Sigma_a) = id_{\perp^0} :\perp^0 \to \perp^0$, so that does not satisfy the functorial requirement because database instance $\alpha^*(\mathcal{A})$ is not isomorphic to $\perp^0$.

2 An integrity constraint $\varphi_i : \mathcal{C}_i \to \mathcal{A}_\emptyset$ (an auxiliary arrow in $\mathbf{Sch}(G)$ obtained from some mapping between two database schemas in G) with $C_i = \alpha^*(\mathcal{C}_i)$ is not isomorphic to $\perp^0$, is not satisfied by α so that, from Definition 5, $\alpha^*(\varphi_i) = id_{\perp^0} :\perp^0 \to \perp^0$, so that does not satisfy the functorial requirement (because database instance $\alpha^*(\mathcal{C}_i)$ is not isomorphic to $\perp^0$). □

Notice that in this functorial semantics for database mappings, an original schema database mapping $\mathcal{M} : \mathcal{A} \to \mathcal{B}$ (with a correspondent arrow $f_\mathcal{M} = \gamma(\mathcal{M}) : \gamma(\mathcal{A}) \to \gamma(\mathcal{B})$, where $\gamma(\mathcal{A})$ and $\gamma(\mathcal{B})$ are the original database schemas enlarged by a number of auxiliary relations introduced in Definition 7) can be translated into the arrow $f = \alpha^*(\gamma(\mathcal{M})) : \alpha^*(\mathcal{A}) \to \alpha^*(\mathcal{B})$ between the instances of the original schema databases $\mathcal{A}$ and $\mathcal{B}$ without added auxiliary relations. This is because $\alpha^*(\mathcal{A})$ and $\alpha^*(\mathcal{A})$ are isomorphic in **DB** to $\alpha^*(\gamma(\mathcal{A}))$ and $\alpha^*(\gamma(\mathcal{A}))$ respectively as follows:

Proposition 6: For any database schemas $\mathcal{A}$ and $\mathcal{B}$, in a given schema database mapping graph G, $A = \alpha^*(\mathcal{A}) \simeq \alpha^*(\gamma(\mathcal{A}))$. Consequently, any functorial semantics of a given schema database mapping $\mathcal{M} : \mathcal{A} \to \mathcal{B}$ is represented in the **DB** category by the morphism, $f \approx \alpha^*(\gamma(\mathcal{M})) : \alpha^*(\mathcal{A}) \to \alpha^*(\mathcal{B})$.

Proof: It is easy to show that $T(\alpha^*(\mathcal{A})) = T(\alpha^*(\gamma(\mathcal{A})))$ because each γ-added relation $r_i(\mathbf{x}_i)$ is just a subrelation of the view obtained by the query $q_{B_i}(\mathbf{x}_i)$ over the relations in the original database $\mathcal{B}$ that is a part of the view mapping $q_{A_i}(\mathbf{x}_i) \Rightarrow q_{B_i}(\mathbf{x}_i) \in \mathcal{M} : \mathcal{A} \to \mathcal{B}$ (see point 2 in Definition 7).

Consequently, we have the isomorphisms $is_A : \alpha^*(\mathcal{A}) \simeq \alpha^*(\gamma(\mathcal{A}))$ and $is_B : \alpha^*(\mathcal{B}) \simeq \alpha^*(\gamma(\mathcal{B}))$ and hence $f = is_B^{-1} \circ \alpha^*(\gamma(\mathcal{M})) \circ is_A : A \to B$, i.e., $f \approx \alpha^*(\gamma(\mathcal{M}))$. □

It is the reason why we can use the original database schemas $\mathcal{A}$ and their database instances $A = \alpha^*(\mathcal{A})$, instead of $\gamma(\mathcal{A})$, in the **DB** category.

4 Conclusions

In this research we defined a base database category **DB** where objects are instance-databases and morphisms between them are extensional GLAV mappings between databases. We defined equivalent (categorically isomorphic) objects (database instances) from the *behavioural point of view based on observations*. That is, each arrow (morphism) is composed of a number of ‘queries’ (view-maps) and each query may be seen as an *observation* over some database instance (object of **DB**). Thus, we characterised each object in **DB** (a database instance) by its behaviour according to a given set of observations. In this way, two databases A and B are equivalent (bisimilar) if they have the same set of observable internal states, i.e., when TA is equal to TB.

We have shown that such a **DB** category is equal to its dual and is symmetric in the way that the semantics of each morphism is a closed object (database) and, vice versa, that each database can be represented by its identity morphism and hence **DB** is a 2-category.

The algebraic database lattice and the categorial (functors) semantics for two basic database operations, *matching* and *merging*, have been introduced in Majkić (2009a).

In the current research, we considered the *schema* level for databases and their view-based mappings, based on queries. The fundamental operations for databases, in the view of inter-mappings between them, are the separation and federation of databases. The inter-mappings depend on the kind of DBMS system used for two mapped databases. When two databases are federated then we can compute the queries over the relations of the both databases. When they are separated by two independent DBMSs then DBMS can compute only the queries with all relations for only one of these two databases.

We have shown that the two fundamental operators, data separation and data federation, used in schema database mapping system, need a different base category from **Set** where coproducts are equal to products (up to isomorphism). Then we defined the graphs schema database mapping systems and the sketches for such database graphs. Consequently we defined the categorical functorial semantics for these sketches into new base database category **DB**.

References

Alagić, S. and Bernstein, P. (2002) 'A model theory for generic schema management', *DBPL, LNCS*, Vol. 2397, pp.228–246, Spinger-Verlag, Berlin.

Barr, M. and Wells, C. (1985) *Toposes, Triples and Theories*, Grundelehren der math. Wissenschaften, Vol. 278, Springer-Verlag, New York, USA.

Barr, M. and Wells, C. (1988) 'The formal description of data types using sketches', in *Mathematical Foundations of Programming Language Semantics, LNCS*, Vol. 298, Springer-Verlag.

Davidson, S., Buneman, P. and Kosky, A. (1998) 'Semantics of database transformations', in B. Thalheim and L. Libkin (Eds.): *Semantics in Databases, LNCS*, Vol. 1358, pp.55–91.

Diskin, Z. and Cadish, B. (1995) 'Algebraic graph-based approach to management of multibase systems: schema integration via sketches and equations', *Proc. 2nd Int. Workshop on Next Generation Information Technologies and Systems (NGITS '95)*, Naharia, Israel, pp.69–79.

Diskin, Z. (1997) *Generalized Sketches as an Algebraic Graph-based Framework for Semantic Modeling and Database Design*, Laboratory for Database Design, FIS/LDBD-97-03, May.

Ehresmann, C. (1966) *Introduction to the Theory of Structured Categories*, Technical Report 10, University of Kansas.

Fagin, R. (1982) 'Horn clauses and database dependencies', *Journal of the ACM*, Vol. 29, No. 5, pp.952–985.

Freyd, P.J. and Kelly, G.M. (1972) 'Categories of continuous functors', *Int. J. of Pure Appl. Algebra*, Vol. 2, No. 3, pp.169–191.

Johnson, M., Rosebrugh, R. and Wood, R.J. (2002) 'Entity-relationship-attribute designs and sketches', *Journal Theory and Applications of Categories*, Vol. 10, No. 3, pp 94–112.

Lair, C. (1996) 'Sur le genre d'esquissibilite des categories modelables (accessibles) possedant les produits de deux', *Diagrammes*, Vol. 35, pp.25–52.

Lawvere, F.W. (1963) 'Functorial semantics of algebraic theories', *Proc. Nat. Acad. Sc.*, Vol. 50, pp.869–872.

Lellahi, S.K. and Spyratos, N. (1990) 'Toward a categorical database model supporting structured objects and inheritance', *Int. Workshop in Information Systems*, October, Kiev, USSR.

Mac Lane, S. (1971) *Categories for the Working Mathematician*, Springer-Verlag, New York, USA.

Madhavan, J., Bernstein, P.A., Domingos, P. and Halevy, A.Y. (2002) 'Representing and reasoning about mappings between domain models', in *AAAI/IAAI*, pp.80–86.

Majkić, Z. and Prasad, B. (2010) 'Kleisli category for database mappings', *International Journal of Intelligent Information and Database Systems (IJIIDS)*, Vol. 4, No. 5, pp.509–527.

Majkić, Z. (2003a) *The Category-theoretic Semantics for Database Mappings*, Technical Report 14-03, University 'La Sapienza', Roma, Italy.

Majkić, Z. (2003b) 'Fixpoint semantics for query answering in data integration systems', *AGP03 – 8th Joint Conference on Declarative Programming*, Reggio Calabria, pp.135–146.

Majkić, Z. (2008) 'Abstract database category based on relational-query observations', *International Conference on Theoretical and Mathematical Foundations of Computer Science (TMFCS-08)*, Orlando FL, USA, July 7–9.

Majkić, Z. (2009a) 'Algebraic operators for matching and merging of relational databases', *International Conference in Artificial Intelligence and Pattern Recognition (AIPR-09)*, Orlando FL, USA, July 13–16.

Majkić, Z. (2009b), 'Induction principle in relational database category', *Int. Conference on Theoretical and Mathematical Foundations of Computer Science (TMFCS-09)*, Orlando FL, USA, July 13–16.

Majkić, Z. (2011a) 'Matching, merging and structural properties of data base category', arXiv: 1102.2395v1, 11 February, pp.1–27.

Majkić, Z. (2011b) 'Data base mappings and monads: (co)induction', arXiv: 1102.4769v1, 22 February, pp.1–31.

Makkai, M. and Pare, R. (1989) 'Accessible categories: the foundations of categorical model theory', *Contemporary Mathematics, Amer. Math. Soc.*, Vol. 104.

Makkai, M. (1994) *Generalized Sketches as a Framework for Completeness Theorems*, Technical report, McGill University.

McLeod, D. and Heimbigner, D. (1985) 'A federated architecture for information management', *ACM Transactions on Information System*, Vol. 3, No. 3, pp.253–278.

Melnik, S., Rahm, E. and Bernstein, P.A. (2003) 'Rondo: a programming platform for generic model management', *SIGMOD*, June 9–12, San Diego, CA.

Rosebrugh, R. and Wood, R.J. (1992) 'Relational databases and indexed categories', *Proc. Int. Category Theory Meeting, CMS Conference Proceedings*, Vol. 13, pp.391–407.

Sheth, A.P. and Larson, J.A. (1990) 'Federated database systems for managing distributed, heterogeneous, and autonomous databases', *ACM Computing Surveys*, Vol. 22, No. 3, pp.183–236.