

Data Base Mappings and Theory of Sketches

Zoran Majkić

International Society for Research in Science and Technology
PO Box 2464 Tallahassee, FL 32316 - 2464 USA
majk.1234@yahoo.com,
<http://zoranmajkic.webs.com/>

Abstract. In this paper we will present the two basic operations for database schemas used in database mapping systems (separation and Data Federation), and we will explain why the functorial semantics for database mappings needed a new base category instead of usual **Set** category.

Successively, it is presented 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 we presented functorial semantics for database mapping systems with the new base category **DB**.

1 Introduction

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

This work is an attempt to give a correct solution for a general problem of complex database-mappings and for high level algebra operators for database schemas (separation, Data Federation), preserving the traditional common practice logical language for schema database mapping definitions.

Only a few works considered this general problem [1,2,3,4]. One of them, which uses a category theory [2], is too restrictive: their institutions can be applied only for *inclusion* mappings between databases.

There is a lot of work for sketch-based denotational semantics for databases [5,6,7,8]. But all of them use, as objects of a sketch category, the elements of an ER-scheme of a database (relations, attributes, etc..) and not the *whole* database as a single object, which is what we need in a framework of inter-databases mappings. It was shown in [9] that if we want to progress to more expressive sketches w.r.t. the original Ehresmann's sketches for diagrams with limits and coproducts, by eliminating non-database objects as, for example, cartesian products of attributes or powerset objects, we need *more expressive arrows* for sketch categories (diagram predicates in [9] that are analog to the approach of Makkai in [10]). Obviously, when we progress to a more abstract vision where objects are the (whole) databases, following the approach of Makkai, in the new basic category **DB** for databases, where objects are just the database instances (each object is a set of relations that compose this database instance), we obtained much

more complex arrows. Such arrows are not simple functions, as in the case of base **Set** category, but complex trees (operads) of view-based mappings. In this way, while Ehresmann's approach prefers to deal with few a fixed diagram properties (commutativity, (co)limitness), we enjoy the possibility of setting full relational-algebra signature of diagram properties.

This work is an attempt to give a correct solution for this problem while preserving the traditional common practice logical language for the schema database mapping definitions.

The *instance level* base database category **DB** has been introduced first time in Technical report [11], and used also in [12]. General information about categories the reader can find in classic books [13], while more information about this particular database category **DB**, with set of its objects Ob_{DB} and set of its morphisms Mor_{DB} , are recently presented in [14]. In this paper we will only emphasize some of basic properties of this **DB** category, in order to render more selfcontained this presentation.

Every object (denoted by A, B, C, \dots) of this category is a database instance, composed by a set of n -ary relations $a_i \in A, i = 1, 2, \dots$ called also "elements of A ".

We consider the views as a universal property for databases: they are the possible observations of the information contained in an instance-database, and we may use them in order to establish an equivalence relation between databases.

In [11] has been defined the power-view operator T , with domain and codomain equal to the set of all database instances, such that for any object (database) A , the object TA denotes a database composed by the set of *all views* of A . The object TA , for a given database instance A , corresponds to the quotient-term algebra \mathbb{L}_A/\approx , where carrier is a set of equivalence classes of closed terms of a well defined formulae of a relational algebra, "constructed" by Σ_R -constructors (relational operators in SPJRU algebra: select, project, join and union) and symbols (attributes of relations) of a database instance A , and constants of attribute-domains.

Different properties of the base **DB** category are considered in a number of previously published papers [15,16,17,18,19] as well, where this basic power-view operator T is extended to the endofunctor $T : \mathbf{DB} \rightarrow \mathbf{DB}$.

The connection between a logical (schema) level and this computational category is based on the *interpretation* functors. Thus, each rule-based conjunctive query at schema level over a database \mathcal{A} will be 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)), are defined usually at a schema level (π_1, π_2 denote first and second projections, \sqcup disjoint union, and \mathcal{N} the set of natural numbers), 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 $r \in R$, $ar : R \rightarrow \mathcal{N}$, with finite arity (finite list of attributes $\mathbf{x} = \langle x_1, \dots, x_n \rangle, n = ar(r) \geq 1$), disjoint from a countable infinite set **att** of attributes (for any single attribute $x \in \mathbf{att}$ a domain of x is a nonempty 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 by \mathbb{S} the set of all database schemas for a given (also infinite) set R .

We denote by \mathcal{A}_\emptyset the empty database schema (where $\pi_1(\mathcal{A}_\emptyset)$ and $\pi_2(\mathcal{A}_\emptyset)$ are empty sets). A *finite* database schema \mathcal{A} is composed by 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 $q(\mathbf{x}) \leftarrow R_1(\mathbf{u}_1), \dots, R_n(\mathbf{u}_n)$, where $n \geq 0$, R_i are 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} , \mathbf{u}_i are free tuples (i.e., may use either variables or constants). Recall that if $\mathbf{v} = (v_1, \dots, v_m)$ then $R(\mathbf{v})$ is a shorthand for $R(v_1, \dots, v_m)$. Finally, each variable occurring in \mathbf{x} must also occur at least once in $\mathbf{u}_1, \dots, \mathbf{u}_n$. Rule-based conjunctive queries (called rules) are composed by: a subexpression $R_1(\mathbf{u}_1), \dots, R_n(\mathbf{u}_n)$, that is the *body*, and $q(\mathbf{x})$ that is the *head* of this rule. If one can find values for the variables of the rule, such that the body holds (i.e. is logically satisfied), then one may deduce the head-fact. This concept is captured by a notion of "valuation". In the rest of this paper a deduced head-fact will be called "a resulting *view* of a query $q(\mathbf{x})$ defined over a database \mathcal{A} ", and denoted by $\|q(\mathbf{x})\|$. Recall that the conjunctive queries are monotonic and satisfiable. 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 a set $\mathcal{M} = \{q_{Ai}(\mathbf{x}_i) \Rightarrow q_{Bi}(\mathbf{x}_i) \mid 1 \leq i \leq n\}$, where \Rightarrow is the logic implication between these conjunctive queries $q_{Ai}(\mathbf{x}_i)$ and $q_{Bi}(\mathbf{x}_i)$, over 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 than it is called a "finitary view" (a finitary view can have also 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 $a_i \in A$ (called also "element of A "). Thus, a relational instance-database A is a set of n -ary relations, and they are managed by relational database systems (RDBMS).

Given two autonomous instance-databases A and B , we can make a *federation* of them, i.e., their disjoint union $A \uplus B$, 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 decentralized. 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.

McLeod and Heimbigner [20] were among the first to define a federated database system, as one which "define[s] the architecture and interconnect[s] databases that minimize central authority yet support partial sharing and coordination among database systems". Among other surveys, Sheth and Larsen [21] define a Federated Database as a collection of cooperating component systems which are autonomous and are possibly heterogeneous.

1.2 DB (Database) category

Based on an observational point of view for relational databases, we may introduce a category **DB** [14] 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, \dots) of this category is a instance-database, composed by a set of n -ary relations $a_i \in A, i = 1, 2, \dots$ called also "elements of A ". We define a universal database instance \mathcal{T} as the union of all database instances, i.e., $\mathcal{T} = \{a_i | a_i \in A, A \in Ob_{DB}\}$. It is the top object of this category. We have that $\mathcal{T} = T\mathcal{T}$, because every view $v \in T\mathcal{T}$ is an instance-database as well, thus $v \in \mathcal{T}$. Vice versa, every element $r \in \mathcal{T}$ is a view of \mathcal{T} as well, thus $r \in T\mathcal{T}$. Every object (instance-database) A has also the empty relation \perp . The object composed by only this empty relation is denoted by \perp^0 and we have that $T\perp^0 = \perp^0 = \{\perp\}$. 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*, as they will be defined by formalism of operads in what follows.

In what follows, the objects in **DB** (i.e., instance-databases) will be called simply databases as well, when it is clear from the context. Each atomic mapping (morphism) in **DB** between two databases is generally composed of three components: the first correspond to conjunctive query q_i over a source database that defines this view-based mapping, the second (optional) w_i "translate" 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 which contribution of this mappings is given to the target relation, i.e., a kind of Global-or-Local-As-View (GLAV) mapping (sound, complete or exact).

In what follows we will consider more simple case without the component w_i .

We introduce also the two functions ∂_0, ∂_1 such that $\partial_0(q_{A_i}) = \{r_{i1}, \dots, r_{ik}\}$ (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-mappings based on operads:

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

- LOGIC-SENTENCE MAPPING: For any sentence, a logic formula φ_i without free variables over a Database schema \mathcal{A} , we can define a schema mapping $\varphi_i : \mathcal{A} \longrightarrow \mathcal{A}_\emptyset$. The unique instance-database of the empty shema \mathcal{A}_\emptyset is denoted by $\perp^0 = \{\perp\}$, where \perp denotes the empty relation. Consequently, for each interpretation α it holds that $\alpha^*(\mathcal{A}_\emptyset) = \perp^0$.
This kind of schema mappings will be used for the integrity constraints over database schemas, and Yes/No queries, as will be 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 $q_i \in O(r_{i1}, \dots, r_{ik}, r_i)$, $Q = (r_{i1}, \dots, r_{ik}) \subseteq \mathcal{A}$.
For a given α the correspondent view-map at instance level is $q_{A_i} = \{\alpha(q_i), q_\perp\} : A \longrightarrow TA$, 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 will 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 \hat{q}_i .

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

Definition 2. ATOMIC MORPHISM: Every schema mapping $f_{Sch} : \mathcal{A} \longrightarrow \mathcal{B}$, based on a set of query-mappings q_i , is defined for finite natural number N by
 $f_{Sch} \triangleq \{v_i \cdot q_i \mid q_i \in O(r_{i1}, \dots, r_{ik}, r'_i), v_i \in O(r'_i, r_i), \{r_{i1}, \dots, r_{ik}\} \subseteq \mathcal{A}, r_i \in \mathcal{B}, 1 \leq i \leq N\}$.

Its correspondent complete morphism at instance database level is

$f = \alpha^*(f_{Sch}) \triangleq \{q_{A_i} = \alpha(v_i) \cdot \alpha(q_i) \mid v_i \cdot q_i \in f_{Sch}\} : A \rightarrow B$, 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, when $\alpha(r'_i) \subseteq \pi_{q_i}(\alpha(r_i))$. Then for any tuple $t \in \alpha(r'_i)$, $\alpha(v_i)(t) = t_1$, for some $t_1 \in \alpha(r_i)$ such that $\pi_{q_i}(\{t_1\}) = t$.
2. exact case, special inclusion case when $\alpha(r'_i) = \pi_{q_i}(\alpha(r_i))$.

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

Notice that the components $\alpha(v_i), \alpha(q_i)$ 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"), and by *c-arrow* we denote the set of all complete morphisms.

Based on atomic morphisms (sets of view-mappings) which are complete arrows (c-arrows), we obtain that their composition generates tree-structures, which can be incomplete (p-arrows), in the way that for a composed arrow $h = g \circ f : A \rightarrow C$, of two atomic arrows $f : A \rightarrow B$ and $g : B \rightarrow C$, we can have the situations where $\partial_0(f) \subset \partial_0(h)$, where 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:

$p\text{-arrow} := c\text{-arrow} \mid c\text{-arrow} \circ c\text{-arrow}$ (for any two c-arrows $f : A \rightarrow B$ and $g : B \rightarrow C$)

$morphism := p\text{-arrow} \mid c\text{-arrow} \circ p\text{-arrow}$ (for any p-arrow $f : A \rightarrow B$ and c-arrow $g : B \rightarrow C$)

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

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

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

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

1. for atomic morphism, $\tilde{f} = B_T(f) \triangleq T\{\|f_i\| \mid f_i \in f\}$.
2. Let $g : A \rightarrow B$ be a morphism with a flux \tilde{g} , and $f : B \rightarrow C$ an atomic morphism with flux \tilde{f} defined in point 1, then $\widetilde{f \circ g} = B_T(f \circ g) \triangleq \tilde{f} \cap \tilde{g}$.

We introduce an equivalence relation over morphisms by, $f \approx g$ iff $\tilde{f} = \tilde{g}$.

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

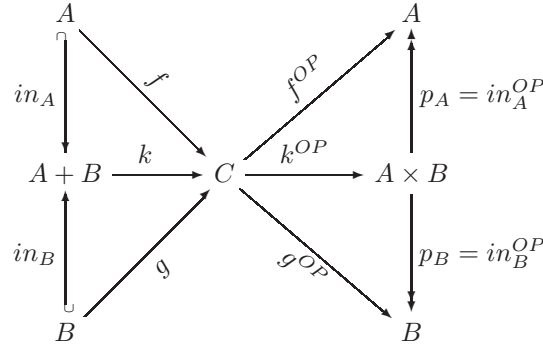
Basic properties of this database category **DB** as its symmetry (bijective correspondence between arrows and objects, duality (**DB** is equal to its dual \mathbf{DB}^{OP}) so that each limit is also colimit (ex. product is also coproduct, pullback is also pushout, empty database \perp^0 is zero object, that is, both initial and terminal object, etc..), and that it is a 2-category has been demonstrated in [11,14].

Generally, database mappings are not simply programs from values (relations) into computations (views) but an equivalence of computations: because of that 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 "internalized" in **DB** category as "computations" [19].

The product $A \times B$ of a databases A and B is equal to their coproduct $A + B$, and the

semantics for them is that we are not able to 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 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:



In the paper [15,16,15] have been considered some relationships of **DB** and standard **Set** category, and has been introduced the categorial (functors) semantics for two basic database operations: *matching* \otimes , and *merging* \oplus , such that for any two databases A and B , we have that $A \otimes B = TA \cap TB$ and $A \oplus B = T(A \cup B)$. In the same work has been defined the algebraic database lattice and has been shown that **DB** is concrete, small and locally finitely presentable (lfp) category. Moreover, it was shown that **DB** is also V-category enriched over itself, was developed a metric space and a subobject classifier for this category, and demonstrated that it is a weak monoidal topos.

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

The plan of this paper is the following: In Section 2 we will present the two basic operations for database schemas used in database mapping systems (separation and federation), and we will explain why the functorial semantics for database mappings needed a new base category instead of common **Set** category.

Finally, in Section 3 is presented a definition of the graph G for a schema database mapping system, and the definition of its sketch category **Sch**(G). Based on this framework then is presented functorial semantics for database mapping systems with the base category **DB**.

2 Basic database schema operations: Separation and Federation

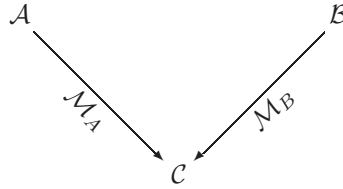
For the composition of complex database mapping graphs, it is important to distinguish two basic compositions of two database schemas \mathcal{A} and \mathcal{B} with respect to DBMSs:

- the case when in this composed schema, the two database schemas are mutually *separated* by two independent DBMSs, in order that it is impossible to write a query over this composition with relations of both databases: it is common case when

two databases are separated, and this symmetric binary separation-composition at schema level will be denoted by $\mathcal{A} \dagger \mathcal{B}$, such that $\pi_i(\mathcal{A} \dagger \mathcal{B}) = \pi_i(\mathcal{A}) \uplus \pi_i(\mathcal{B}), i = 1, 2$.

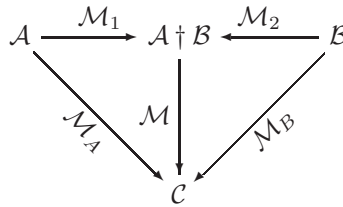
- the case when in this composed schema, the two database schemas are *connected* into the same DBMS (without any change of the two original database schemas): In this case we are able to use the queries over this composed schema with relations of *both* databases for inter database mappings, and this symmetric binary federation-composition at schema level will be denoted by $\mathcal{A} \oplus \mathcal{B}$, such that $\pi_i(\mathcal{A} \oplus \mathcal{B}) = \pi_i(\mathcal{A}) \uplus \pi_i(\mathcal{B}), i = 1, 2$.

The identity $=$ for database schemas is naturally defined by: for any two $\mathcal{A}, \mathcal{B} \in \mathbb{S}$, $\mathcal{A} = \mathcal{B}$ if $\pi_i(\mathcal{A}) = \pi_i(\mathcal{B}), i = 1, 2$. Notice that both symmetric binary operators, \dagger, \oplus , 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 $((\mathbb{S}, =), \oplus, \mathcal{A}_\emptyset)$ are the monoids. Let us consider, for example, a mapping $\mathcal{M} : \mathcal{A} \dagger \mathcal{B} \rightarrow \mathcal{C}$, and a mapping $\mathcal{M} : \mathcal{A} \oplus \mathcal{B} \rightarrow \mathcal{C}$. In the first case in any query mapping $q(\mathbf{x}) \Rightarrow q_C(\mathbf{x}) \in \mathcal{M}$ the 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 equivalently represent by the graph:



where $\mathcal{M}_A \uplus \mathcal{M}_B = \mathcal{M}$, while in the case of mapping $\mathcal{M} : \mathcal{A} \oplus \mathcal{B} \rightarrow \mathcal{C}$ such an 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} .

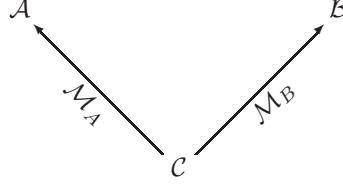
If we introduce the mappings $\mathcal{M}_1 = \{r_{Ai}(\mathbf{x}_i) \Rightarrow r_{Ai}(\mathbf{x}_i) | r_{Ai} \in \mathcal{A}\}$ and $\mathcal{M}_2 = \{r_{Bi}(\mathbf{y}_i) \Rightarrow r_{Bi}(\mathbf{y}_i) | r_{Bi} \in \mathcal{B}\}$, then we obtain the mapping graph,



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

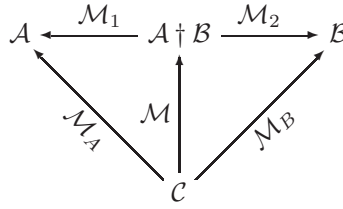
Let us consider another dual example, a mapping $\mathcal{M} : \mathcal{C} \rightarrow \mathcal{A} \dagger \mathcal{B}$. In this case in any query mapping $q_C(\mathbf{x}) \Rightarrow q(\mathbf{x}) \in \mathcal{M}$ the all relation symbols in the query $q(\mathbf{x})$ must be of database \mathcal{A}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

equivalently represent by the graph:



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

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,



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*. The detailed presentation of sketches for the database mappings, and their functorial semantics will be given in Section 3. Sketches are developed by Ehresmann's school, especially by R.Guitartand and C.Lair [22,23,24]. Sketch is a category together with 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. Given any sketch, one can consider the underlying graph of the sketch as a (many-sorted) language, and one can write down axioms in the $L_{\infty,\infty}$ -logic (the infinitary FOL with finite quantifiers) over this language, so that the models of the axioms become exactly the models of the sketch.

the category of models of a given sketch has as objects the models, and arrows 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 G_0 and a set of arrows denoted G_1 together with the operators $dom, cod : G_1 \rightarrow G_0$ which assigns to each arrow its source and target. (Co)cones and diagrams are defined for graphs in exactly the same way as they are for categories, but commutative co(cones) and diagrams of course make no sense for graphs.

By a sketch me mean 4-tuple (G, u, D, C) where G is a graph, $u : G_0 \rightarrow G_1$ 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 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 , 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 a such framework we do not use commutative database mapping systems, thus D is empty set. In fact, in the database mapping system, the (co)cone diagrams above will never be used in practical representation of database mapping systems: instead will be alternatively used only its selfconsistent parts, as first diagram above, or, equivalently, a single arrow $\mathcal{M} : \mathcal{A} \dagger \mathcal{B} \rightarrow \mathcal{C}$.

But, clearly, for the introduced dataschema composition operator \dagger , with cone and cocone diagrams above has to be present 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 \mathbf{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 $F : \mathbf{E} \rightarrow \mathbb{B}$, where \mathbb{B} is a base category. Then all cones in C has to be functorially translated into limit commutative diagrams in \mathbb{B} , and all cocones in C has 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 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} and $\times, +$ the product and coproduct operators in \mathbb{B} , but it 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, so that does not hold the isomorphism $A \times B \simeq A + B$ in **Set**.

□

Remark: The fundamental consequence of this lemma is that we needed 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, so that 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, so that it is impossible to compute the queries with the relations from both databases.*

The disjoint property for mappings is represented by facts that

$$\partial_0(f + g) \triangleq \partial_0(f) + \partial_0(g), \quad \partial_1(f + g) \triangleq \partial_1(f) + \partial_1(g).$$

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 we have that $T(A + B) = TA + TB$. Consequently $A + A$ is not isomorphic to A .*

Proof: We have that $T(A + B) = TA + TB$, directly from 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, holds that $T(\widetilde{f + g}) = T(\widetilde{f} + \widetilde{g}) = T\widetilde{f} + T\widetilde{g} = \widetilde{f} + \widetilde{g} = \widetilde{f + g}$.

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

□

Notice that for coproducts holds that $C + \perp^0 = \perp^0 + C \simeq C$, 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, such that $\partial_0(\perp^1) = \partial_1(\perp^1) = \perp^0$, with $\widetilde{\perp^1} = \perp^0$.

We are ready now 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 $in_B : B \hookrightarrow A + B$.*

*By duality property we have that **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) $p_A = in_A^{OP} : A \times B \twoheadrightarrow A$ and $p_B = in_B^{OP} : A \times B \twoheadrightarrow B$, 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, id_B are the identity arrows of A and B respectively, holds that $\widetilde{id_A + id_B} = \widetilde{id_A} + \widetilde{id_B} = TA + TB = T(A + B) = \widetilde{id_{A+B}}$. 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 given $k : A \longrightarrow A_1$, $k_1 : A_1 \longrightarrow A_2$, $l : B \longrightarrow B_1$, $l_1 : B_1 \longrightarrow B_2$, holds $+^1(k_1, l_1) \circ +^1(k, l) = +^1(k_1, l_1) \cap +^1(k, l) = k_1 \circ k + l_1 \circ l = +^1(k_1 \circ k, l_1 \circ l) = +^1((k_1, k) \circ (l_1, l))$, 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 ($in_A = TA$, $in_B = TB$).

It is easy to verify that for any two arrows $f : A \longrightarrow C$, $g : B \longrightarrow C$, there is 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 .

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 decentralized. 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.

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 *disjoint union* of them under the same DBMS, thus, equal to database $A \uplus B$.

3 Categorical 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* (*tg*d) and *equality-generating dependency* (*eg*d). We denote by \mathcal{A}_\emptyset the empty database schema with empty set of relation symbols, where $\Sigma_{\mathcal{A}_\emptyset}$ is the empty set of integrity constraints.

A *tg*d says 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 *eg*d says that if some tuples, satisfying certain equalities exist in the relation, then some values in these tuples must be equal. Functional dependencies are *eg*d's of a special form, as for example primary-key integrity constraints.

These two classes of dependencies together comprise the *embedded implication dependencies* (EID) [25] which seem to include essentially all of the naturally-occurring constraints on relational databases (the bolded variables \mathbf{x}, \mathbf{y} denotes a nonempty list of variables):

1. a *tuple-generating dependency* (*tg*d) of the FOL form

$$\forall \mathbf{x} (\exists \mathbf{y} \phi_A(\mathbf{x}, \mathbf{y}) \Rightarrow \exists \mathbf{z} (\psi_A(\mathbf{x}, \mathbf{z})))$$
 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 will consider only class of weakly-full *tg*d 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 side).
2. an *equality-generating dependency* (*eg*d): $\forall \mathbf{x} (\phi_A(\mathbf{x}) \Rightarrow (\mathbf{x}_1 = \mathbf{x}_2))$
 where a formula $\phi_A(\mathbf{x})$ is a conjunction of atomic formulas over \mathcal{A} , and $\mathbf{x}_1, \mathbf{x}_2$ 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 $\text{tg}\mathbf{d} \ \forall \mathbf{x} (\exists \mathbf{y} \ \phi_A(\mathbf{x}, \mathbf{y}) \Rightarrow \exists \mathbf{z} (\psi_B(\mathbf{x}, \mathbf{z})))$, that is by the *view mapping* $q_A(\mathbf{x}) \Rightarrow q_B(\mathbf{x})$, as will be used in what follows in Definition 7, 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 $(\exists \mathbf{z} \ \psi_B(\mathbf{x}, \mathbf{z}))$), is a query over the schema \mathcal{B} .

In what follows we will 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 chapter. The integrity constraints for databases are expressed by the FOL logical sentences (the FOL formulae without free variables), and 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}_\emptyset . We define their denotation in the **DB** category as follows:

Definition 5. *For any sentence $\varphi : \mathcal{A} \rightarrow \mathcal{A}_\emptyset$ (a logic 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, $f : \mathcal{A} \rightarrow \perp^0$, 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 , 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, differently from view-mappings for queries (formulae with free variables) given in Definition 2, the integrity constraints have in a **DB** category the empty database (zero object, i.e., terminal and initial) as the codomain, and the information flux equal to $\perp^0 = \{\perp\}$. It is consistent with definition of morphisms in **DB** category, because the sentences do not transfer any data from source to target database, so, 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 this semantics for logic 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 will use two fundamental operators for the database schemas, data federation \oplus and data separation \dagger , with the two correspondent monoids, $((\mathbb{S}, =), \dagger, \mathcal{A}_\emptyset)$ and $((\mathbb{S}, =), \oplus, \mathcal{A}_\emptyset)$, and with the distribution law: $\mathcal{A} \oplus (\mathcal{B} \dagger \mathcal{C}) = (\mathcal{A} \oplus \mathcal{B}) \dagger (\mathcal{A} \oplus \mathcal{C})$.

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

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

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

is an atomic schema in the schema expression \mathcal{A} .

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

Proposition 3 *For a given interpretation α the following homomorphism from schema database level and instance database level there exists:*

$$\alpha^* : ((\mathbb{S}, =), \oplus, \dagger, \mathcal{A}_\emptyset) \rightarrow ((Ob_{DB}, \simeq), \uplus, +, \perp^0).$$

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**, while for every interpretation $\alpha^*(\mathcal{A}_\emptyset) = \perp^0$.

From the monoidal property we have the equation $\mathcal{A} \oplus \mathcal{A}_\emptyset = \mathcal{A}$ in the algebra \mathbb{S}_{Alg} . By the homomorphism above we have that $\alpha^*(=) = \simeq, \alpha^*(\oplus) = \uplus$, so that $\alpha^*(\mathcal{A} \oplus \mathcal{A}_\emptyset) = \alpha^*(\mathcal{A}) \uplus \alpha^*(\mathcal{A}_\emptyset) = A \uplus \perp^0 \simeq A$. 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 that $\alpha^*(\dagger) = +$, so that $\alpha^*(\mathcal{A} \dagger \mathcal{A}_\emptyset) = \alpha^*(\mathcal{A}) + \alpha^*(\mathcal{A}_\emptyset) = A + \perp^0$, and for coproducts in **DB** it holds the isomorphism $A + \perp^0 \simeq A$.

□

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 $\Sigma_A = \Sigma_A^{tgd} \cup \Sigma_A^{egd} = \pi_2(\mathcal{A})$ are the database integrity constraints (set of EIDs) which can be empty st as well.

We can represent it by a sketch schema mapping $\phi_A : \mathcal{A} \rightarrow \mathcal{A}_\emptyset$ (ϕ_A denotes the sentence obtained by conjunction of all formulae in Σ_A), where \mathcal{A}_\emptyset is the empty schema, such that for any interpretation α it holds that $\alpha(\mathcal{A}_\emptyset) = \perp^0$.

Proposition 4 *If for a database schema $\mathcal{A} = (S_A, \Sigma_A)$ there exists a model A which satisfies all integrity constraints $\Sigma_A = \Sigma_A^{tgd} \cup \Sigma_A^{egd}$ (ϕ_A will denote 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 $\alpha^* : \mathbf{Sch}(G) \rightarrow \mathbf{DB}$, where $\mathbf{Sch}(G)$ is the sketch category derived from the graph G with the arrow $\Sigma_A : \mathcal{A} \rightarrow \mathcal{A}_\emptyset$ (i.e., $\mathbf{Sch}(G)$ is composed by the objects $\mathcal{A}, \mathcal{A}_\emptyset$, the arrow $\phi_A : \mathcal{A} \rightarrow \mathcal{A}_\emptyset$ and the identity arrows $id_{\mathcal{A}} : \mathcal{A} \rightarrow \mathcal{A}$ and $id_{\mathcal{A}_\emptyset} : \mathcal{A}_\emptyset \rightarrow \mathcal{A}_\emptyset$), such that:*

1. $\alpha^*(\mathcal{A}) \triangleq A$, where A is possibly empty database instance (with all empty relations) as well
2. $\alpha^*(id_{\mathcal{A}}) \triangleq id_A : A \rightarrow A$
3. $\alpha^*(id_{\mathcal{A}_\emptyset}) \triangleq id_{\perp^0} : \perp^0 \rightarrow \perp^0$
4. $\alpha^*(\phi_A) \triangleq (f_{tgd} \cup f_{egd}) : A \rightarrow \perp^0$.

Proof: from Definition 5 and the point 4 of this proposition we have that 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 nonempty database $A = \alpha^*(\mathcal{A})$, that is an object in the **DB** category.

□

Notice that any empty database A (such that 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, based on the fact that any arrow for this empty database $f : A \rightarrow A$ has the information flux $\tilde{f} = \perp^0$, so that $f = id_A$ is the unique identity arrow for this empty database. But the unique arrows $g : A \rightarrow \perp^0$ and $h : \perp^0 \rightarrow A$ have the same information fluxes, i.e., $\tilde{g} = \tilde{h} = \perp^0$, so that $g \circ h = id_{\perp^0}$ and $h \circ g = id_A$, and, consequently, $A \simeq \perp^0$. Consequently, the remark in the point 1 of this Proposition specifies that if $A = \alpha^*(\mathcal{A}) \simeq \perp^0$ is empty database, then α^* is a model of a schema \mathcal{A} , and integrity constraint for point 4 corresponds to the satisfaction of this integrity constraint w.r.t. the Definition 5.

3.2 Categorical semantics of database mappings

First of all, we will define formally a schema database mapping graph G , as follows:

Definition 6. A schema database mapping graph G is composed by 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 \mathcal{A} and \mathcal{B} . we will 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 will denote their sequential composition in this graph G by $\mathcal{M}_2 ; \mathcal{M}_1$, where $;$ is a binary associative but non commutative operator.
- Given two mappings $\mathcal{M}_1 : \mathcal{A} \rightarrow \mathcal{B}$ and $\mathcal{M}_2 : \mathcal{A} \rightarrow \mathcal{C}$, we will denote their branching in this graph G by $\mathcal{M}_2 \uplus \mathcal{M}_1 : \mathcal{A} \rightarrow \mathcal{B} \uplus \mathcal{C}$, where \uplus is a binary associative and commutative operator of disjoint union.

This 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} , is a constraint on the pairs of interpretations, of \mathcal{A} and \mathcal{B} , and therefore specifies which pairs of interpretations can co-exist, given the mapping (see also [1]). The formalization 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 will use the same symbol for this γ -enlarged database schema by these new relations). Then we introduce in $\mathbf{Sch}(G)$ the single mapping arrow $f_{\mathcal{M}} : \mathcal{A} \rightarrow \mathcal{B}$, where,

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

where q_i, 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:

We introduce in this sketch $\mathbf{Sch}(G)$ 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 $\partial_0(w_i) = \partial_1(q_i)$, $\partial_1(w_i) = \{c_i\}$), and $f_{BC_i} = w'_i \cdot q'_i : \mathcal{B} \rightarrow \mathcal{C}_i$ (with $\partial_0(q'_i) = R'_i$, $\partial_1(q'_i) = \partial_0(w'_i)$, $\partial_1(w'_i) = \{c_i\}$, 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 $\{q_{Ai}(\mathbf{x}_i) \Rightarrow c_i(\mathbf{x}_i, \mathfrak{h}_A)\}$ and f_{BC_i} corresponds to $\{q_{Bi}(\mathbf{x}_i) \Rightarrow c_i(\mathbf{x}_i, \mathfrak{h}_B)\}$, where $\mathfrak{h}_A, \mathfrak{h}_B$ are two new values not present in the domain of the databases. Consequently, we introduce in G also the integrity constraint arrow $\varphi_i : \mathcal{C}_i \rightarrow \mathcal{A}_\emptyset$ for this new schema \mathcal{C}_i , where the sentence φ_i is equal to the tgd $\forall \mathbf{x}_i (\exists y (c_i(\mathbf{x}_i, y) \wedge y = \mathfrak{h}_A) \implies \exists z (c_i(\mathbf{x}_i, z) \wedge z = \mathfrak{h}_B))$.

It is easy to verify that in the obtained sketch $\mathbf{Sch}(G)$, between given any two nodes there is at maximum *one* arrow. There is a fundamental functorial *interpretation* connection from schema mappings and their models in the instance level category **DB**: based on the Lawvere categorial theories [26,27], where he introduced a way of describing algebraic structures using categories for theories, functors (into base category **Set**, which we will substitute by more adequate category **DB**), and natural transformations for morphisms between models.

For example, Lawvere's seminal observation that the theory of groups is a category with group object, that group in **Set** is a product preserving functor, and that a morphism of groups is a natural transformation of functors, is an original new idea that was successively extended in order 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 by 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 [28]. 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 **Set**^E by Ehresmann-Kennison theorem. A generalization to base categories other than **Set** was proved by Freyd and Kelly (1972). In what follows we will 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 $\mathbf{Sch}(G)$) is presented in the left commutative diagram below (notice that mapping arrow \mathcal{M} in a graph G are 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:

$$\begin{array}{ccc}
 \gamma(\mathcal{A}) & \xrightarrow{f_{\mathcal{M}_1}} \gamma(\mathcal{A}) \uparrow \gamma(\mathcal{B}) & \xleftarrow{f_{\mathcal{M}_2}} \gamma(\mathcal{B}) \\
 & \searrow f_{\mathcal{M}_A} \quad \downarrow f_{\mathcal{M}} \quad \swarrow f_{\mathcal{M}_B} & \\
 & \gamma(\mathcal{C}) &
 \end{array}
 \quad \Rightarrow \quad
 \begin{array}{ccc}
 A & \xrightarrow{In_A} A + B & \xleftarrow{In_B} B \\
 & \searrow j \quad \downarrow k \quad \swarrow i & \\
 & C &
 \end{array}$$

As we explained in the introduction, in database mapping systems, expressed by a graph G , we will never use "commutative diagrams" as left diagram above (but only an arrow $f_{\mathcal{M}} : \gamma(\mathcal{A}) \uparrow \gamma(\mathcal{B}) \rightarrow \gamma(\mathcal{C})$, or, more frequently, two simple arrows $f_{\mathcal{M}_A} = \gamma(\mathcal{M}_A) : \gamma(\mathcal{A}) \rightarrow \gamma(\mathcal{C})$ and $f_{\mathcal{M}_B} = \gamma(\mathcal{M}_B) : \gamma(\mathcal{B}) \rightarrow \gamma(\mathcal{C})$), our sketch $\mathbf{E} = \mathbf{Sch}(G)$ will be 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 , obtained by applying 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 $f_{Sch} : \mathcal{A} \rightarrow \mathcal{B}$, (where \mathcal{B} is not empty schema) the atomic morphism in **DB** category $\alpha^*(f_{Sch}) : \alpha^*(\mathcal{A}) \rightarrow \alpha^*(\mathcal{B})$ is determined by banal set-inclusion case of Definition 2.*

Then there is the functor (categorical model) $\alpha^ : \mathbf{Sch}(G) \rightarrow \mathbf{DB}$. The set of categorical 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** will be called a DB-mapping system, and denoted by \mathcal{M}_S .*

Proof: This is easy to verify, based on general theory for sketches [27]: 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** (labeled tree without any interpretation). The functor α^* is only the 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 and for two atomic mappings $f_{Sch} : \mathcal{A} \rightarrow \mathcal{B}$, $g_{Sch} : \mathcal{B} \rightarrow \mathcal{C}$, and their atomic morphisms in **DB**, $f = \alpha^*(f_{Sch})$, $g = \alpha^*(g_{Sch})$, 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 $\varphi_i : \mathcal{C}_i \rightarrow \mathcal{A}_\emptyset$ (in Definition 7) the operator α^* satisfies the functorial property, such that this schema arrow is mapped into the arrow $\alpha^*(\varphi_i) : C_i \rightarrow \perp^0$, where $C_i = \alpha^*(\mathcal{C}_i)$. In fact it holds, 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 the Definition 5 the functorial property is satisfied.

Notice that this is true also in special cases when $C_i = \alpha^*(\mathcal{C}_i) \simeq \perp^0$. This case happens 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 empty relation in the database instance $C_i = \alpha^*(\mathcal{C}_i) = \{\alpha(c_i(\mathbf{x}, y))\}$, so that $C_i \simeq \perp^0$.

Thus an integrity constraint $\varphi_i : \mathcal{C}_i \rightarrow \mathcal{A}_\emptyset$ (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 can not be a functor from $\mathbf{Sch}(G)$ into \mathbf{DB} . In order that a given α is not a model of G , must be satisfied one of the following cases:

1. case when 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 $\Sigma_A : \mathcal{A} \rightarrow \mathcal{A}_\emptyset$ 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, it holds that $\alpha^*(\Sigma_a) = id_{\perp^0} : \perp^0 \rightarrow \perp^0$, and it does not satisfy the functorial requirement because database instance $\alpha^*(\mathcal{A})$ is not isomorphic to \perp^0 .
2. case when some integrity constraint $\varphi_i : \mathcal{C}_i \rightarrow \mathcal{A}_\emptyset$ (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 it holds that $\alpha^*(\varphi_i) = id_{\perp^0} : \perp^0 \rightarrow \perp^0$, and it 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 from an original schema database mapping $\mathcal{M} : \mathcal{A} \rightarrow \mathcal{B}$, with a correspondent arrow $f_{\mathcal{M}} = \gamma(\mathcal{M}) : \gamma(\mathcal{A}) \rightarrow \gamma(\mathcal{B})$, where $\gamma(\mathcal{A}), \gamma(\mathcal{B})$ are the original database schemas enlarged by a number of auxiliary relations introduced in Definition 7, can be translated into the arrow $f = \alpha^*(\gamma(\mathcal{M})) : \alpha^*(\mathcal{A}) \rightarrow \alpha^*(\mathcal{B})$ between the instances of the original schema databases \mathcal{A} and \mathcal{B} without added auxiliary relations, because we have that $\alpha^*(\mathcal{A}), \alpha^*(\mathcal{B})$ are isomorphic in \mathbf{DB} to $\alpha^*(\gamma(\mathcal{A})), \alpha^*(\gamma(\mathcal{B}))$, respectively, as follows:

Proposition 6 *For any database schema \mathcal{A} and \mathcal{B} , in a given schema database mapping graph G , it holds that $A = \alpha^*(\mathcal{A}) \simeq \alpha^*(\gamma(\mathcal{A}))$.*

Consequently, any functorial semantics of a given schema database mapping $\mathcal{M} : \mathcal{A} \rightarrow \mathcal{B}$ is represented in the \mathbf{DB} category by the morphism, $f \approx \alpha^(\gamma(\mathcal{M})) : \alpha^*(\mathcal{A}) \rightarrow \alpha^*(\mathcal{B})$.*

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 part of the view mapping $q_{A_i}(\mathbf{x}_i) \Rightarrow q_{B_i}(\mathbf{x}_i) \in \mathcal{M} : \mathcal{A} \rightarrow \mathcal{B}$ (see point 2 in Definition 7).

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

□

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

4 Conclusions

In previous work 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 *behavioral point of view based on observations*: each arrow (morphism) is composed by a number of "queries" (view-maps), and each query may be seen as an *observation* over some database instance (object of **DB**). Thus, we characterized each object in **DB** (a database instance) by its behavior 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 its observable internal states, i.e. when TA is equal to TB . It has been shown that such a **DB** category is equal to its dual, it is symmetric in the way that the semantics of each morphism is an closed object (database) and viceversa each database can be represented by its identity morphism, so that **DB** is a 2-category.

In [15,15] has been introduced the categorial (functors) semantics for two basic database operations: *matching* and *merging* (and data federation), and has been defined the algebraic database lattice.

Here 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 is the fact if they are separated or federated databases. It depends 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 both databases; when they are separated by two independent DBMS, then DBMS can compute only the queries with all relations of only one of these two databases.

We have shown that these 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 categorial functorial semantics for these sketches into new base database category **DB**.

References

1. J.Madhavan, P.A.Bernstein, P.Domingos, and A.Y.Halevy, "Representing and reasoning about mappings between domain models," *In AAAI/IAAI*, pp. 80–86, 2002.
2. S.Alagić and P.Bernstein, "A model theory for generic schema management," *DBPL 2001, LNCS 2397, Springer-Verlag, Berlin*, pp. 228–246, 2002.
3. S.Davidson, P.Buneman, and A.Kosky, "Semantics of database transformations," *In B.Thalheim, L.Libkin, Eds. Semantics in Databases, LNCS 1358*, pp. 55–91, 1998.
4. S.Melnik, E.Rahm, and P.A.Bernstein, "Rondo: A programming platform for generic model management," *SIGMOD 2003, June 9-12, San Diego, CA*, 2003.

5. S.K.Lellahi and N.Spyratos, "Toward a categorical database model supporting structured objects and inheritance," *Int. Workshop in Information Systems 90*, October, Kiev, USSR, 1990.
6. R.Rosebrugh and R.J.Wood, "Relational databases and indexed categories," *Proc. Int. Category Theory Meeting, CMS Conference Proceedings 13*, pp. 391–407, 1992.
7. Z.Diskin and B.Cadish, "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, 1995.
8. M.Johnson and R.Rosebrugh, "Entity-relationship models and sketches," *Journal Theory and Applications of Categories*, 2000.
9. Z.Diskin, "Generalized sketches as an algebraic graph-based framework for semantic modeling and database design," *Laboratory for Database Design, FIS/LDBD-97-03*, May, 1997.
10. M.Makkai, "Generalized sketches as a framework for completeness theorems," *Technical report, McGill University*, 1994.
11. Z. Majkić, "The category-theoretic semantics for database mappings," *Technical Report 14-03, University 'La Sapienza', Roma, Italy*, 2003.
12. Z.Majkić, "Fixpoint semantics for query answering in data integration systems," *AGP03 - 8.th Joint Conference on Declarative Programming, Reggio Calabria*, pp. 135–146, 2003.
13. S. Mac Lane, *Categories for the Working Mathematician*, Springer-Verlag, 1971.
14. Z.Majkić, "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, 2008.
15. Z.Majkić, "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, 2009.
16. Z.Majkić, "Induction principle in relational database category," *Int. Conference on Theoretical and Mathematical Foundations of Computer Science (TMFCS-09)*, Orlando FL, USA, July 13-16, 2009.
17. Z.Majkić, "Matching, merging and structural properties of data base category," *arXiv: 1102.2395v1*, 11 February, pp. 1–27, 2011.
18. Z.Majkić, "Data base mappings and monads: (co)induction," *arXiv: 1102.4769v1*, 22 February, pp. 1–31, 2011.
19. Z.Majkić and B.Prasad, "Kleisli category for database mappings," *International Journal of Intelligent Information and Database Systems (IJIIDS)*, Volume 4, Number 5, pp. 509–527, 2010.
20. Amit P. Sheth and James A. Larson, "Federated database systems for managing distributed, heterogeneous, and autonomous databases," *ACM Computing Surveys*, vol. 22, no. 3, pp. 183–236, 1990.
21. D.McLeod and D.Heimbigner, "A Federated architecture for information management," *ACM Transactions on Information System*, Vol.3, Issue 3, pp. 253–278, 1985.
22. C.Ehresmann, "Introduction to the theory of structured categories," *Technical Report 10, University of Kansas*, 1966.
23. C.Lair, "Sur le genre d'esquissibilité des catégories modelables (accessibles) possédant les produits de deux," *Diagrammes* 35, pp. 25–52, 1996.
24. M.Barr and C.Wells, "The formal description of data types using sketches," *In Mathematical Foundations of Programming Language Semantics*, vol 298, LNCS, Springer-Verlag, 1988.
25. R.Fagin, "Horn clauses and database dependencies," *Journal of the ACM*, vol. 29, pp. 952–985, 1982.
26. F.W.Lawvere, "Functorial semantics of algebraic theories," *Proc. Nat. Acad. Sc.* 50, pp. 869–872, 1963.

27. M.Barr and C.Wells, "Toposes, Triples and Theories," *Grundlehren der math. Wissenschaften* 278, Springer-Verlag, 1985.
28. M.Makkai and R.Pare, "Accessible categories: the foundations of categorical model theory," *Contemporary Mathematics* 104, Amer. Math. Soc., 1989.