Abstract. We present a categorical denotational semantics for a database mapping, based on views, in the most general framework of a database integration/exchange. Developed database category $DB$, for databases (objects) and view-based mappings (morphisms) between them, is different from $Set$ category: the morphisms (based on a set of complex query computations) are not functions, while the objects are database instances (sets of relations). The logic-based schema mappings between databases, usually written in a highly expressive logical language (ex. LAV, GAV, GLAV mappings, or tuple generating dependency) may be functorially translated into this "computation" category $DB$. A new approach is adopted, based on the behavioral point of view for databases, and behavioral equivalences for databases and their mappings are established. By introduction of view-based observations for databases, which are computations without side-effects, we define a fundamental (Universal algebra) monad with a power-view endofunctor $T$. The resulting 2-category $DB$ is symmetric, so that any mapping can be represented as an object (database instance) as well, where a higher-level mapping between mappings is a 2-cell morphism. Database category $DB$ has the following properties: it is equal to its dual, complete and cocomplete. Special attention is devoted to practical examples: a query definition, a query rewriting in GAV Database-integration environment, and the fixpoint solution of a canonical data-integration model.

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. Today, this ‘global’ approach cannot be avoided because of the necessity of P2P open-ended networks of heterogeneous databases. The aim of this work is a definition of category $DB$ for database mappings more suitable than a $Set$ category: The databases are more complex structures w.r.t. sets, and the mappings between them are too complex to be represented by a single (complete) function. Why do we need an enriched categorical semantic domain such as this for databases? We will try to give a simple answer to this question:
- 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 databases (merging,
matching, etc.), preserving the traditional common practice logical language for schema
database mapping definitions.

- The query-rewriting algorithms are not integral parts of a database theory (used
to define a database schema with integrity constraints); they are programs and we need
an enriched context that is able to formally express these programs through mappings
between databases as well.

- Let us consider, for example, P2P systems or mappings in a complex Dataware-
house: formally, we would like to make a synthetic graphic representations of database
mappings and queries and to develop a graphic tool for a meta-mapping description of
complex (and partial) mappings in various contexts, with a formal mathematical back-
ground.

Only a few works considered this general problem [1–4]. 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 [5–8].
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 this 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 will obtain much
more complex arrows, as we will see. 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 defini-
tions. Different properties of this DB category are considered in a number of previously

This paper follows the following plan: In Section 2 we present an Abstract Object Type
based on view-based observations. In Section 3 we develop a formal definition for a
Database category $DB$, its power-view endofunctor, and its duality property. In Section
4 we formulate the two equivalence relations for databases (objects in $DB$ category):
a strong and a weak observation equivalences. Finally, in Section 5 we present an ap-
lication of this theory to the data integration/exchange systems, with an example for
a query-rewriting in data integration system, and we define a fixpoint operator for an
infinite canonical solution in data integration/exchange systems.

2
1.1 Technical Preliminaries

The database mappings, for a given logical language, are defined usually at a schema level, as follows:

- A database schema is a pair $\mathcal{A} = (S_h, S_n)$ where: $S_h$ is a countable set of relation symbols $r \in R$ with finite arity, disjoint from a countable infinite set $\text{att}$ of attributes (for any $x \in \text{att}$ a domain of $x$ is a nonempty subset $\text{dom}(x)$ of a countable set of individual symbols $\text{dom}$, disjoint from $\text{att}$), such that for any $r \in R$, the sort of $r$ is a finite sequence of elements of $\text{att}$. $S_n$ denotes a set of closed formulas called integrity constraints, of the sorted first-order language with sorts $\text{att}$, constant symbols $\text{dom}$, relational symbols $R$, and no function symbols. A finite database schema is composed by a finite set $S_h$, so that the set of all attributes of such a database is finite.

- An instance of a database $\mathcal{A}$ is given by $\mathcal{A} = (\mathcal{A}, I_A)$, where $I_A$ is an interpretation function that maps each schema element of $S_h$ (n-ary predicate) into an n-ary relation $a_i \in \mathcal{A}$ (called also "element of $\mathcal{A}$"). Thus, a relational instance-database $\mathcal{A}$ is a set of n-ary relations.

- We consider a rule-based conjunctive query over a database schema $\mathcal{A}$ as an expression $q(x) \leftarrow R_1(u_1), ..., R_n(u_n)$, 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 $\mathcal{A}, u_i$ are free tuples (i.e., may use either variables or constants). Recall that if $v = (v_1, ..., v_m)$ then $R(v)$ is a shorthand for $R(v_1, ..., v_m)$. Finally, each variable occurring in $x$ must also occur at least once in $u_1, ..., u_n$. Rule-based conjunctive queries (called rules) are composed by: a subexpression $R_1(u_1), ..., R_n(u_n)$, that is the body, and $q(x)$ 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 $q(x)$ defined over a database $\mathcal{A}$". 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 databases $\mathcal{A}$ and $\mathcal{B}$ is expressed by an union of "conjunctive queries with the same head". Such mappings are called "view-based mappings". Consequently we consider a view of an instance-database $\mathcal{A}$ an n-ary relation (set of tuples) obtained by a "select-project-join + union" (SPJRU) query $q(x)$ (it is a term of SPJRU algebra) over $\mathcal{A}$: 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).

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. Database category $DB$, which will be introduced in what follows, is at an instance level, i.e., any object in $DB$ is an instance-database (i.e., a set of relations). 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 $\mathcal{A}$.
(a model of the database schema $A$) to the instance-database composed by all views of $A$.

In what follows we will work with the typed operads, first developed for a purpose of homotopy theory [16–18], having a set $R$ of types (each relation symbol is a type), or “$R$-operads” for short. The basic idea of an $R$-operad $O$ is that, given types $r_1, \ldots, r_k, r \in R$, there is a set $O(r_1, \ldots, r_k, r)$ of abstract $k$-ary “operations” with inputs of type $r_1, \ldots, r_k$ and output of type $r$. We can visualize such an operation as a tree with only one node. In an operad, we can obtain new operations from old ones by composing them: it can be visualized in terms of trees (Fig. 1) We can obtain the new operators from old ones by permuting arguments, and there is a unary “identity” operation of each type. Finally, we insist on a few plausible axioms: the identity operations act as identities for composition, permuting arguments is compatible with composition, and composition is associative. Thus, formally, we have the following:

**Definition 1.** For any set $R$, an $R$-operad $O$ consists of

1. for any $r_1, \ldots, r_k, r \in R$, a set $O(r_1, \ldots, r_k, r)$
2. for any $f \in O(r_1, \ldots, r_k, r)$ and any $g_1 \in O(r_11, \ldots, r_{1i1}, r_1), \ldots, g_k \in O(r_{k1}, \ldots, r_{k1k}, r_k)$, an element $f \cdot (g_1, \ldots, g_k) \in O(r_{11}, \ldots, r_{1k1}, \ldots, r_{k1}, \ldots, r_{k1k}, r)$
3. for any $r \in O$, an element $1_r \in O(r, r)$
4. for any permutation $\sigma \in R_k$, a map $\sigma : O(r_1, \ldots, r_k, r) \to O(r_{\sigma(1)}, \ldots, r_{\sigma(k)}, r)$, $f \mapsto f \sigma$, such that:
   (a) whenever both sides make sense, $f \cdot (g_1 \cdot (h_{11}, \ldots, h_{1i1}), \ldots, g_k \cdot (h_{k1}, \ldots, h_{ki1})) = (f \cdot (g_1, \ldots, g_k)) \cdot ((h_{11}, \ldots, h_{1i1}), \ldots, (h_{k1}, \ldots, h_{ki1}))$
   (b) for any $f \in O(r_1, \ldots, r_k, r)$, $f = 1_r \cdot f = f \cdot (1_{r_1}, \ldots, 1_{r_k})$
   (c) for any $f \in O(r_1, \ldots, r_k, r)$, and $\sigma, \sigma_1 \in R_k$. $f(\sigma \sigma_1) = (f \sigma) \sigma_1$
   (d) for any $f \in O(r_1, \ldots, r_k, r)$, $\sigma \in R_k$ and $g_1 \in O(r_{11}, \ldots, r_{1i1}, r_1), \ldots, g_k \in O(r_{k1}, \ldots, r_{k1k}, r_k)$, $(f \sigma) \cdot ((g_{\sigma(1)}, \ldots, g_{\sigma(k)})) = (f \cdot (g_1, \ldots, g_k)) \rho(\sigma)$
   where $\rho : R_k \to R_{1+\ldots+i_k}$ is the obvious homomorphism.
(e) for any \( f \in O(r_1, \ldots, r_k, r) \), \( g_1 \in O(r_{i_1}, \ldots, r_{i_2}, r_1) \), \( g_k \in O(r_{k_1}, \ldots, r_{k_2}, r_k) \),
and \( \sigma_1 \in R_{i_1}, \ldots, \sigma_k \in R_{i_k} \),
\( (f \cdot (g_1 \sigma_1, \ldots, g_k \sigma_k)) = (f \cdot (g_1, \ldots, g_k)) \cdot g_1(\sigma_1, \ldots, \sigma_k) \),
where \( g_1 : R_{i_1} \times \ldots \times R_{i_k} \longrightarrow R_{i_1 + \ldots + i_k} \) is the obvious homomorphism.

Let us define the "R-algebra" of an operad where its abstract operations are represented by actual functions (query-functions). For a given database schema with relation symbols \( r_1, \ldots, r_k \) we consider \( f \in O(r_1, \ldots, r_k, r) \) as a conjunctive query \( r \leftarrow r_1, \ldots, r_k \)
that defines a view \( r \).

**Definition 2.** For any R-operad \( O \), a R-algebra \( \alpha \) consists of:

1. For any \( r \in R \), a set \( \alpha(r) \) is a set of tuples of this type (relation).
2. For any \( q \in O(r_1, \ldots, r_k, r) \) a mapping function \( \alpha(q) : \alpha(r_1) \times \ldots \times \alpha(r_k) \longrightarrow \alpha(r) \),
   such that
   (a) whenever both sides make sense, \( \alpha(q \cdot (q_1, \ldots, q_k)) = \alpha(q) \cdot (\alpha(q_1) \times \ldots \times \alpha(q_k)) \)
   (b) for any \( r \in R \), \( \alpha(1_r) \) acts as an identity on \( \alpha(r) \)
   (c) for any \( q \in O(r_1, \ldots, r_k, r) \) and a permutation \( \sigma \in R_k \), \( \alpha(q \sigma) = \alpha(q) \sigma \),
   where \( \sigma \) acts on the function \( \alpha(q) \) on the right by permuting its arguments.
3. We introduce the two functions, \( \partial_0 \) and \( \partial_1 \), such that for any \( \alpha(q), q \in O(r_1, \ldots, r_k, r) \),
   we have that \( \partial_0(q) = \{r_1, \ldots, r_k\} \), \( \partial_0(\alpha(q)) = \{\alpha(r_1), \ldots, \alpha(r_k)\} \), \( \partial_1(q) = \{r\} \),
   and \( \partial_1(\alpha(q)) = \{\alpha(r)\} \).

Consequently, we can think of an operad as a simple sort of theory, used to define a schema mappings between databases, and its algebras as models of this theory used to define the mappings between instance-databases, where a mapping \( \alpha \) is considered as an interpretation of relation symbols of a given database schema.

## 2 Data Object Type for query-answering database systems

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 can use them in order to establish an equivalence relation between databases.

In a theory of algebraic specifications an Abstract Data Type (ADT) is specified by a set of operations (constructors) that determine how the values of the carrier set are built up, and by a set of formulae (in the simplest case, equations) stating which values should be identified. In the standard initial semantics, the defining equations impose a congruence on the initial algebra. Dually, a coalgebraic specification of a class of systems, i.e., Abstract Object Types (AOT), is characterized by a set of operations (destructors) that specify what can be observed out of a system-state (i.e., an element of the carrier), and how a state can be transformed to a successor-state.

We start by introducing the class of coalgebras for database query-answering systems for a given instance-database (a set of relations) \( A \). They are presented in an algebraic style, by providing a co-signature. In particular, the sorts include one single "hidden sort" corresponding to the carrier of the coalgebra, and other "visible" sorts, for inputs and outputs, that have a given fixed interpretation. Visible sorts will be interpreted as sets without any algebraic structure defined on them. For us, coalgebraic terms, built
only over destructors, are precisely interpreted as the basic observations that one can make on the states of a coalgebra.

Input sorts are considered as a set $L_A$ of union of conjunctive queries $q(x)$ for a given database $A$, where $x$ is a tuple of variables (attributes) of this query. Each query has an algebraic term of the “select-project-join + union” algebraic query language (SPJRU, or equivalent to it, SPCU algebra, Chapter 4.5, 5.4 in [?]) with a carrier equal to the set of relations in $A$. We define the power view-operator $T$, with domain and codomain equal to the set of all instance-databases, 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 instance-database $A$, corresponds to the quotient-term algebra $L_A/\approx$, where the carrier is a set of equivalence classes of closed terms of a well-defined formulae of a relational algebra. Such formulae are "constructed" by $\Sigma_R$-constructors (relational operators in SPJRU algebra: select, project, join and union), by symbols (attributes of relations) of a database instance $A$, and by constants of attribute-domains. More precisely, $TA$ is "generated" by this quotient-term algebra $L_A/\approx$. For every object $A$ holds that $A \subseteq TA$, and $TA = TTA$, i.e., each (element) view of database instance $TA$ is also an element (view) of a database instance $A$. Notice that when $A$ is also finitary (has a finite number of relations) but with at least one relation with infinite number of tuples, then $TA$ has an infinite number of relations (views of $A$), thus can be an infinitary object. It is obvious that when a domain of constants of a database is finite then both $A$ and $TA$ are finitary objects. As default we assume that a domain of every database is an arbitrary large finite set. This is a reasonable assumption for real applications.

Consequently, the output sort of this database $AOT$ is a set $TA$ of all resulting views (resulting n-ary relation) obtained by computation of queries $q(x)$ over a database $A$. It is considered as the carrier of a coalgebra as well.

**Definition 3.** A co-signature for a Database query-answering system, for a given instance-database $A$, is a triple $\Sigma = (S, OP, [\_])$, where $S$ are the sorts, $OP$ are the operators, and $[\_]$ is an interpretation of visible sorts, such that:

1. $S = (X_A, L_A, T)$, where $X_A$ is a hidden sort (a set of states of a database $A$), $L_A$ is an input sort (set of union of conjunctive queries), and $T$ is an output sort (the set of all views of all instance-databases).

2. $OP$ is a set of operations: a method $\text{Next} : X_A \times L_A \rightarrow X_A$, that corresponds to an execution of a next query $q(x) \in L_A$ in a current state of a database $A$, such that a database $A$ passes to the next state; and $\text{Out} : X_A \times L_A \rightarrow TA$ is an attribute that returns with the obtained view of a database for a given query $q(x) \in L_A$.

3. $[\_]$ is a function, mapping each visible sort to a non-empty set.

The Data Object Type for a query-answering system is given by a coalgebra: $\lambda N_{\text{ext}}, \lambda \text{Out} > : X_A \rightarrow X_A^{L_A} \times T A^{L_A}$, of the polynomial endofunctor $(\_)^{L_A} \times T A^{L_A} : \text{Set} \rightarrow \text{Set}$, where $\lambda$ denotes the lambda abstraction for functions of two variables into functions of one variable (here $Z^Y$ denotes the set of all functions from $Y$ to $Z$).

This separation between the sorts and their interpretations is given in order to obtain a conceptual clarity: we will simply ignore it in the following by denoting both, a sort and the corresponding set, by the same symbol. In an object-oriented terminology, the coalgebras are expressive enough in order to specify the parametric methods and the
attributes for a database (conjunctive) query answering systems. In a transition system terminology, such coalgebras can model a deterministic, non-terminating, transition system with inputs and outputs. In [19] a complete equational calculus for such coalgebras of restricted class of polynomial functors has been defined.

In the rest of this paper we will consider only the database query-answering systems without side effects: that is, the obtained results (views) will not be materialized as a new relation of this database $A$. Thus, when a database answers a query, it remains in the same initial state. Thus, the set $X_A$ is a singleton $\{A\}$ for a given database $A$, and consequently it is isomorphic to the terminal object 1 in the Set category. As a consequence, from $1^{\mathcal{L}_A} \simeq 1$, we obtain that a method $\text{Next}$ is just an identity function $id : 1 \rightarrow 1$. Consequently, the only interesting part of this AOT, is the attribute part $Out : X_A \times \mathcal{L}_A \rightarrow TA$, with the fact that $X_A \times \mathcal{L}_A = \{A\} \times \mathcal{L}_A \simeq \mathcal{L}_A$.

Consequently, we obtain an attribute mapping $Out : \mathcal{L}_A \rightarrow TA$, which will be used as a semantic foundation for a definition of database mappings: for any query $q_i(x) \in \mathcal{L}_A$, the correspondent algebraic term $\tilde{q}_i$ is a function (it is not a T-coalgebra) $\tilde{q}_i : A^k \rightarrow TA$, where $A^k$ is k-th cartesian product of $A$ and $r_{i1}, \ldots, r_{ik} \in A$ are the relations used for computation of this query. A view-mapping can be defined now as a T-coalgebra $q_A : A \rightarrow TA$, that, obviously, is not a function. We introduce also the two functions $\delta_0, \delta_1$ such that $\delta_0(q_A) = \{r_{i1}, \ldots, r_{ik}\}$ and $\delta_1(q_A) = \{r_i\}$, with obtained view $r_i = \|q_i(x)\| = \tilde{q}_i(r_{i1}, \ldots, r_{ik})$. Thus, we can formally introduce a theory for operads:

**Definition 4.** VIEW-MAPPING: For any query over a schema $\mathcal{A}$ we can define a schema map $q_i : \mathcal{A} \rightarrow TA$, where $q_i \in O(r_{i1}, \ldots, r_{ik}, r_i)$, $Q = \{r_{i1}, \ldots, r_{ik}\} \subseteq \mathcal{A}$, and $r_i \in TA$.

A correspondent view-map at instance level is $q_A = \{q_i, \alpha_A\} : \mathcal{A} \rightarrow TA$, with $A = \alpha^*(\mathcal{A})$, $TA = \alpha^*(TA)$, $\delta_0(q_A) = \delta_1(q_A) = \{\perp\}$. 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, $\delta_0(q_A) = \alpha^*(Q) \subseteq \mathcal{A}$ and $\delta_1(q_A) = \alpha(r) \subseteq TA$ is a singleton with the unique element equal to view obtained by a "select-project-join+union" term $\tilde{q}_i$.

3 Database category DB

Based on an observational point of view for relational databases, we may introduce a category $DB$ [20] 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 a instance-database, composed by a set of n-ary relations $a_i \in A$, $i = 1, 2, \ldots$ 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.

A closed object in $DB$ is a instance-database $A$ such that $A = TA$. We have that $\mathcal{T} = TT$, because every view $v \in TT$ 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 $r \in TT$. 

Every object (instance-database) $A$ has also an empty relation $\bot$. The object composed by only this empty relation is denoted by $\bot^0$ and we have that $T \bot^0 = \bot^0 = \{ \bot \}$. Any empty database (a database with only empty relations) is isomorphic to this bottom object $\bot^0$.

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).

Instead of lists $(g_1, ..., g_k)$ used for mappings in Definitions 1, 2, we will use the sets $\{g_1, ..., g_k\}$ because a mapping between two databases does not depend on a particular permutation of its components. Thus, we introduce an atomic morphism (mapping) between two databases as a set of simple view-mappings:

**Definition 5.** Atomic Morphism: Every schema mapping $f_{Sch} : A \rightarrow B$, based on a set of query-mappings $q_i$, is defined for finite natural number $N$ by

$$f_{Sch} \equiv \{ v_i \cdot w_i \cdot q_i \mid q_i \in O(r_{i1}, ..., r_{ik}), w_i \in O(r_{i}^p, r_{i}), v_i \in O(r_{i}^p, r_{i}) $$

$$\{r_{i1}, ..., r_{ik}\} \subseteq A, r_i \in B, 1 \leq i \leq N\}$$

Its correspondent complete morphism at instance database level is

$$f = \alpha^*(f_{Sch}) \equiv \{ q_{A_i} = \alpha(v_i) \cdot \alpha(w_i) \cdot \alpha(q_i) \mid v_i \cdot w_i \cdot q_i \in f_{Sch}\} : A \rightarrow B.$$ Each $\alpha(q_i)$ is a query computation, with obtained view $\alpha(r_{i}^p) \in TA$ for an instance-database $A = \alpha^*(A) = \{r_{ik} \mid r_k \in A\}$, and $B = \alpha^*(B)$.

Each $\alpha(v_i) : \alpha(r_{i}^p) \rightarrow \alpha(r_{i}^p)$, where $\alpha(r_i^p) \in TB$, is equal to the function determined by the symmetric domain relation $r_{AB} \subseteq dom_A \times dom_B$ for the equivalent constants in $\alpha^*(A)$ and $\alpha^*(B)$ $(a, b) \in r_{AB}$ means that, $a \in dom_A$ and $b \in dom_B$ represent the same entity of the real word (requested for a federated database environment) as: for any $(a_1, ..., a_n) \in \alpha(r^p)$ holds $\alpha(w_i)(a_1, ..., a_n) = (b_1, ..., b_n)$, and for all $1 \leq k \leq n \ (a_k, b_k) \in r_{AB}$. If $r_{AB}$ is not defined, it is assumed, by default, that $\alpha(w_i)$ is an identity function.

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

1. inclusion case, when $\alpha(r_i^p) \subseteq P_{q_i}(\alpha(r_i))$. Then for any tuple $t \in \alpha(r_i^p)$, $\alpha(v_i)(t) = t_1$, for some $t_1 \in \alpha(r_i)$ such that $P_{q_i}(\{t_1\}) = t$.

We define $\|q_{A_i}\| \equiv \alpha(r_i^p)$ the extension of data transmitted from an instance-database $A$ into $B$ by a component $q_{A_i}$. 8
2. inverse-inclusion case, when \( \alpha(r_i') \supseteq P_q(\alpha(r_i)) \).

Then, for any tuple \( t \in \alpha(r_i') \),

\[
\alpha(v_i)(t) = \begin{cases} t_1 & \text{if } \exists t_1 \in \alpha(r_i) \cdot P_q(\{t_1\}) = t \\ \text{empty tuple, otherwise} & \end{cases}
\]

We define \( \|q_A\| = P_q(\alpha(r_i)) \) the extension of data transmitted from an instance-
database \( A \) into \( B \) by a component \( q_A \).

3. equal case, when both (a) and (b) are valid.

Notice that the components \( \alpha(v_i), \alpha(w_i), \alpha(q_i) \) 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

\( q_A : A \longrightarrow TA \), which is an atomic morphism, is a complete morphism (the case

when \( B = TA, r_{AB} \) is not defined, and \( \alpha(v_i) \) belongs to the "equal case"), and by

c-arrow we denote the set of all complete morphisms.

**Example 1:** In the Local-as-View (LAV) mappings [21], the inverse inclusion, inclusion

and equal case correspond to the sound, complete and exact view respectively. In the

Global-as-View (GAV) mappings, the inverse inclusion, inclusion and equal case
correspond to the complete, sound and exact view respectively.

\( \square \)

**Remark:** In the rest of this paper we will consider only empty domain relations (i.e.,

when \( \alpha(w_i) \) are the identity functions) and we will write \( r \in A \) also for \( \alpha(r) \in \alpha^*(A) \),

i.e., the name (type) of a relation \( r \) in \( A \) is used also for its extension (set of tuples of

that relation), and \( A \) for \( \alpha^*(A) \) as well. Notice that the functions \( \partial_0 \) and \( \partial_1 \) are different

from \( \text{dom} \) and \( \text{cod} \) functions used for the category arrows. Here \( \partial_0 \) specifies exactly

the subset of relations in a database \( A \) used for view-based mapping, while \( \partial_1 \) defines

the target relation in a database \( B \) for this mapping. Thus: \( \partial_0(f) \subseteq \text{dom}(f) = A, \)

\( \partial_1(f) \subseteq \text{cod}(f) = B \) (in the case when \( f \) is a simple view-mapping then \( \partial_1(f) \) is a

singleton). In fact, we have that they are functions \( \partial_0, \partial_1 : \text{Mor}_{DB} \longrightarrow \mathcal{P}(T) \) (where \( \mathcal{P} \)

is the powerset operation), such that for any morphism \( f : A \rightarrow B \) between databases

\( A \) and \( B \), we have that \( \partial_0(f) \subseteq A \) and \( \partial_1(f) \subseteq B \).

The Yes/No query \( q_i \) over a database \( A \), obviously do not transfer any information to

target object \( TA \). Thus, if the answer to such a query is \( Yes \), then this query is repre-

sented in \( DB \) category as a mapping \( q_i : A \rightarrow TA \), such that the source relations in

\( \partial_0(q_i) \) are non-empty and \( \partial_1(q_i) = \{\bot\} \). The answer to such a query \( q_i \) is \( No \) iff (if

and only if) such a mapping does not exist in this \( DB \) category.

\( \square \)

We are ready now to give a formal definition for all morphisms in the category \( DB \).

Generally, a composed morphism \( h : A \rightarrow C \) is a general tree such that all its leaves

are not in \( A \): such a morphism is denominated as an incomplete (or partial) p-arrow.

**Definition 6. Syntax:** The following BNF defines the set of all morphisms in \( DB \):

\[
p - \text{arrow} ::= c - \text{arrow} | c - \text{arrow} \circ c - \text{arrow} \text{ (for any two } \text{c-arrows } f : A \longrightarrow B \text{ and } g : B \longrightarrow C \text{)} \]

\[
morphism ::= p - \text{arrow} | c - \text{arrow} \circ p - \text{arrow} \text{ (for any } p - \text{arrow } f : A \longrightarrow B \text{ and } \text{c-arrow } g : B \longrightarrow C \text{)}
\]
whereby the composition of two arrows, $f$ (incomplete) and $g$ (complete), we obtain the following $p$-arrow $h = g \circ f : A \rightarrow C$

$$h = g \circ f = \bigcup_{q_{B_j} \in \alpha^*(gsch) \& \partial_0(q_{B_j}) \cap \partial_1(f) \neq \emptyset} \{q_{B_j}\} \circ \bigcup_{q_{A_i} \in \alpha^*(fsch) \& \partial_0(q_{A_i}) = \{v\} \& v \in \partial_0(q_{B_j})} \{q_{A_i}(\text{tree})\}$$

$$= \{q_{B_j} \circ \{q_{A_i}(\text{tree}) \mid \partial_1(q_{A_i}) \subseteq \partial_0(q_{B_j})\} \mid q_{B_j} \in \alpha^*(gsch) \& \partial_0(q_{B_j}) \cap \partial_1(f) \neq \emptyset\}$$

$$= \{q_{B_j}(\text{tree}) \mid q_{B_j} \in \alpha^*(gsch) \& \partial_0(q_{B_j}) \cap \partial_1(f) \neq \emptyset\}$$

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

We have the equal analog diagrams of schema mappings as well:

- For a morphism $f : A \rightarrow B$ in $DB$ we have syntactically identical schema mapping arrow $f_{sch} : A \rightarrow B$ without the interpretation of its symbols (the composition of functions “$\circ$” is replaced by the associative composition of operads “$\cdot$”).

- A schema mapping graph $G$ is any subset of schema arrows.

Notice that the arrows (morphisms) in $DB$ are not functions. Thus, $DB$ is different from $Set$ category. In order to explain the composition of morphisms let us consider the following example:

**Example 2**: Let us consider the morphisms $f : A \rightarrow B, g : B \rightarrow C$, such that
Let us see, for example, the composition of the c-arrow $h : C \rightarrow D$ with the composed arrow $g \circ f$ in the previous example, where $D = \{d_1, \ldots, d_4\}$, $h = \{q_{c_1}, q_{c_2}, q_{c_3}\}$, $\partial_0(q_{c_1}) = \{c_1\}$, $\partial_0(q_{c_2}) = \{d_2\}$, $\partial_0(q_{c_3}) = \{c_1\}$, $\partial_1(q_{c_1}) = \{d_1\}$, $\partial_1(q_{c_2}) = \{c_2\}$, $\partial_1(q_{c_3}) = \{c_3\}$, with $q_B(\text{tree}) = q_B \circ \{q_{a_2}, q_{a_3}\}$ a complete, and $q_B(\text{tree}) = q_B \circ \{q_{a_1}, -\}$ a partial (incomplete) component of this tree, as represented in the Fig. 4.

As we see, a composition of (complete) morphisms generally produces a partial (incomplete) morphism (only a part of the tree $h_T$ represents a real contribution from $A$ into $C$) with hidden elements (in the diagram of the composed morphism $h$, the element $b_4$ is a hidden element). In such a representation we "forgot" parts of the tree $g_T \circ f_T$ that are not involved in real information contribution of composed mappings from the source into the target object. So, we define the semantics of any morphism $h : A \rightarrow C$ as an "information transmitted flux" from the source into the target object. An "information flux" (denoted by $\tilde{h}$) is a set of views (so, it is an object in $DB$ category as well) which is "transmitted" by a mapping.

In order to explain this concept of "information flux" let us consider a simple morphism $f : A \rightarrow B$ from a database $A$ into a database $B$, composed by only one view
map based on a single query \( q(x) \leftarrow R_1(u_1), \ldots, R_n(u_n) \), where \( n \geq 0 \), \( R_i \) are relation names (at least one) in \( A \) or built-in predicates (ex. \( \leq, = \), etc..), and \( q \) is a relation name not in \( A \). Then, for any tuple \( c \) for which the body of this query is true, also \( q(c) \) must be true, that is, this tuple from a database \( A \) "is transmitted" by this view-mapping into one relation of database \( B \). The set (n-ary relation) \( Q \) of all tuples that satisfy the body of this query will constitute the whole information "transmitted" by this mapping.

The "information flux" \( \tilde{f} \) of this mapping is the set \( TQ \), that is, the set of all views (possible observations) that can be obtained from the transmitted information of this mapping.

**Definition 7.** 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") that are really "transmitted" from the source to the target object.

1. For an atomic morphism, \( \tilde{f} = B_T(f) \triangleq T\{ \| q_{A_i} \| \mid q_{A_i} \in f \} \).
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 \( f \circ g = B_T(f \circ g) \triangleq \tilde{f} \cap \tilde{g} \).

Thus we have the following fundamental property:

**Proposition 1** Any mapping morphism \( f : A \rightarrow B \) is a closed object in \( DB \), i.e., \( \tilde{f} = T\tilde{f} \).

**Proof:** This proposition may be proved by structural induction; each atomic arrow is a closed object \( (T\tilde{f} = T(T\{ \| q_{A_i} \| \mid q_{A_i} \in f \}) = T\{ \| q_{A_i} \| \mid q_{A_i} \in f \} = \tilde{f} \). Each arrow is a composition of a number of complete arrows, and intersection of closed objects is always a closed object.

□

**Remark:** The "information flux" \( \tilde{f} \) of a given morphism (mapping) \( f : A \rightarrow B \) is an instance-database as well (its elements are the views defined by the formulae above),
thus, an object in $DB$: the minimal “information flux” is equal to the bottom object $\perp^0$
so that, given any two database instances $A, B$ in $DB$, there exists at least an arrow
(morphism) between them $f : A \rightarrow B$ such that $\tilde{f} = \perp^0$.

**Proposition 2** The following properties for morphisms are valid:

1. each arrow $f : A \rightarrow B$, such that $\tilde{f} = TB$ is an epimorphism
2. each monic and epic arrow $f : A \leftarrow B$, such that $\tilde{f} = TA$ is a monomorphism
3. each monic and epic arrow is an isomorphism, thus two objects $A$ and $B$ are iso-

**Proof:** 1. An arrow $f : A \rightarrow B$ is epic iff for any $h, g : B \rightarrow C$ holds ($h \circ f = g \circ f$) \Rightarrow ($h = g$), thus ($\tilde{h} \circ \tilde{f} = \tilde{g} \circ \tilde{f}$) \Rightarrow ($\tilde{h} = \tilde{g}$) which is satisfied by $\tilde{f} = TB$
(because $\tilde{h} \subseteq T B$ and $\tilde{g} \subseteq T B$)
2. An arrow $f : A \rightarrow B$ is monic iff for any $h, g : C \rightarrow A$ holds ($f \circ h = f \circ g$) \Rightarrow
($h = g$), thus ($\tilde{f} \circ \tilde{h} = \tilde{f} \circ \tilde{g}$) \Rightarrow ($\tilde{h} = \tilde{g}$) which is satisfied by $\tilde{f} = TA$
(because $\tilde{h} \subseteq T A$ and $\tilde{g} \subseteq T A$)
3. By 1 and 2, because an isomorphism is epic and monic, and viceversa if $f$ is monic
and epic then $\tilde{f} = TA$ (2) and $\tilde{f} = TB$ (1), thus $TA = TB$. It is enough to show
the isomorphism $A \cong TA$ : let us define the isomorphisms $is_A : A \rightarrow TA$, and its
inverse $is_A^{-1} : TA \rightarrow A$,

$$is_A = \bigcup_{\partial_1(q_A) = \{v\} \& v \in TA} \{q_A\}, \quad is_A^{-1} = \bigcup_{\partial_1(q_{TA}) = \partial_1(q_{TA}), \& v \in A} \{q_{TA}\}$$

Thus, $\tilde{is_A} = \tilde{is_A}^{-1} = TA$, so it holds that $\tilde{is_A}^{-1} \circ \tilde{is_A} = TA = \tilde{id}_A = \tilde{is_A} \circ \tilde{is_A}^{-1}$, i.e.,
$\tilde{is_A}^{-1} \circ \tilde{is_A} = \tilde{id}_A$ and $\tilde{is_A} \circ \tilde{is_A}^{-1} = \tilde{id}_{TA}$, thus $A \cong TA$. Finally, $A \cong TA = TB \cong B$,
i.e., $A \cong B$. \qed

**Remark:** Thus, we consider, for example, the real object (empty database instance) $\perp^0$
as zero object (both terminal and initial) in $DB$, (from any real object $A$ in $DB$ there is
a unique arrow from it into $\perp^0$ and its reversed arrow). Each arrow $f$ with $\partial_1(f) = \{\perp\}$
or $\partial_1(f) = \perp$ has an empty flux, thus does not give any information contribution to the
target database: as for example Yes arrows in $DB$ for Yes/No queries.

It is easy to verify that each empty database (with all empty relations) is isomorphic to
the zero object $\perp^0$.

In what follows we will show that any two isomorphic objects (databases) in $DB$ are
observationally equivalent.

### 3.1 Interpretations of schema mappings

The semantics of mapping between two relational database schemas, $f : A \rightarrow B$, is
a constraint on the pairs of interpretations, of $A$ and $B$, and therefore specifies which
pairs of interpretations can co-exist, given the mapping (see also [1]). We consider only
view-based mappings between schemas defined in the SQL language of SPJRU algebra, i.e., when
(1) \( f = \{ q_{A_1}(x) \Rightarrow b_j(x) \} \), where \( q_{A_1}(x) \) is a union of conjunctive queries over \( A \) and \( b_j \) is a relation symbol of a database schema \( B \), or,
(2) \( f = \{ q_{A_1}(x) \Rightarrow q_{B_1}(x) \} \), where \( q_{B_1}(x) \) is a union of conjunctive queries over \( B \). In this case the mapping \( f \) also involves a helper database schema \( C \) with a relation \( c_i(x) \) for each \( q_{A_1}(x) \in f \) with two new database mappings, \( f_{AC} : A \rightarrow C \) and \( f_{BC} : B \rightarrow C \), with \( f_{AC} = \{ q_{A_1}(x) \Rightarrow c_i(x) \} \) and \( f_{BC} = \{ q_{B_1}(x) \Rightarrow c_i(x) \} \).

The formula \( e = q_{A_1}(x) \Rightarrow q_{B_1}(x) \) (logical implication between queries), means that each tuple of the view obtained by the query \( q_{A_1}(x) \) is also a tuple of the view obtained by the query \( q_{B_1}(x) \).

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 [22, 23], 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 any small sketch \( E \) 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 \( Mod(E) \) is an accessible category by Lair’s theorem and reflexive subcategory of \( Set^E \) by Ehresmann-Kennison theorem. In what follows we will substitute the base category \( Set \) by this new database category \( DB \).

**Proposition 3** Let \( Sch(G) \) be a schema category generated from a schema mapping graph (sketch) \( G \). Every interpretation \( R \)-algebra \( \alpha \) has as its categorial correspondent the functor (categorial model) \( \alpha^* : Sch(G) \rightarrow DB \), defined as follows:

1. for any database schema \( A = \{a_1, \ldots, a_n\} \) (object in \( Sch(G) \)), where \( a_i \in R \), \( i = 1, \ldots, n \), holds \( A \triangleq \{ \alpha(a_1), \ldots, \alpha(a_n) \} \); i.e., \( A \) is an interpretation (logical model) of a database schema \( A \).
2. for any schema mapping arrow \( f : A \rightarrow B \), let \( f_T \) be the tree structure of operads, \( f_T = \{ f_1 \cdot g_1, \ldots, f_k \cdot g_k \} \), where each \( f_i \) is a linear composition of operads, then \( \alpha^*(f) = \{ \alpha(f_1) \circ \alpha^*(g_1), \ldots, \alpha(f_k) \circ \alpha^*(g_k) \} \), otherwise \( \alpha^*(f) = \alpha(f_T) \).

Formally, the satisfaction of mapping \( f \) is defined as follows: for each logical formula \( e \in f \), \( \{ \alpha^*(A), \alpha^*(B) \} \models e \), that is \( e \) is satisfied by a model \( \alpha^* \in Mod(Sch(G)) \subseteq DB^{Sch(G)} \).

**Proof:** This is easy to verify, based on general theory for sketches [23]; each arrow in a sketch (enriched schema mapping graph) \( G \) may be converted into a tree syntax structure of some morphism in \( DB \) (labeled tree without any interpretation), thus, a sketch
$G$ can be extended into a category $\text{Sch}(G)$. (The composition of schema mappings in the category $\text{Sch}(G)$, where each mapping is a set of first-order logical formulas, can be defined as a disjoint union). The functor is only the simple extension of the interpretation $R$-algebra function $\alpha$ for a lists of symbols, as in Definition 5. □

3.2 Power-view endofunctor $T$

Let us extend the notion of the type operator $T$ into a notion of the endofunctor in $DB$ category:

**Theorem 1** There exists an endofunctor $T = (T^0, T^1) : DB \to DB$, such that

1. for any object $A$, the object component $T^0(A)$ is equal to the type operator $T$, i.e., $T^0(A) \triangleq TA$
2. for any morphism $f : A \to B$, the arrow component $T^1$ is defined by

$$T(f) \triangleq T^1(f) = \bigcup_{\partial_0(qTA_i) = \partial_1(qTA_i) = \{v\}} \{v\} \in f$$

3. Endofunctor $T$ preserves the properties of arrows, i.e., if a morphism $f$ has a property $P$ (monic, epic, isomorphic), then also $T(f)$ has the same property: let $P_{\text{mono}}, P_{\text{epi}}$ and $P_{\text{iso}}$ are monomorphic, epimorphic and isomorphic properties respectively, then the following formula is true

$$\forall (f \in \text{Mor}_{DB})(P_{\text{mono}}(f) \equiv P_{\text{mono}}(Tf) \text{ and } P_{\text{epi}}(f) \equiv P_{\text{epi}}(Tf) \text{ and } P_{\text{iso}}(f) \equiv P_{\text{iso}}(Tf)).$$

**Proof:** It is easy to verify that $T$ is a 2-endofunctor and to see that $T$ preserves properties of arrows: for example, if $P_{\text{mono}}(f)$ is true for an arrow $f : A \to B$, then $\hat{T}f = TA$ and $Tf = T\hat{f} = T(TA) = TA$, thus $P_{\text{mono}}(Tf)$ is true. Viceversa, if $P_{\text{mono}}(Tf)$ is true then $Tf = T\hat{f} = T(TA)$, i.e., $\hat{f} = TA$ and, consequently, $P_{\text{mono}}(f)$ is true. □

The endofunctor $T$ is a right and left adjoint to identity functor $I_{DB}$, i.e., $T \simeq I_{DB}$. Thus we have the equivalence adjunction $< T, I_{DB}, \eta, \eta^\circ >$ with the unit $\eta^C : T \to I_{DB}$ (such that for any object $A$ the arrow $\eta^C_A \triangleq \eta^C(A) \equiv \eta_A^{-1} : TA \to A$, and the counit $\eta : I_{DB} \simeq T$ (such that for any $A$ the arrow $\eta_A \triangleq \eta(A) \equiv \eta_A : A \to TA$ are isomorphic in $DB$ (by duality theorem it holds that $\eta^C = \eta^{\text{inv}}$).

The function $T^1 : (A \to B) \to (TA \to TB)$ is not a higher-order function (arrows in $DB$ are not functions): thus, there is no correspondent monad-comprehension for the monad $T$, which invalidates the thesis [25] that "monads $\equiv$ monad-comprehensions". It is only valid that "monad-comprehensions $\Rightarrow$ monads".

We have already seen that the views of a database can be seen as its observable computations: what we need, to obtain an expressive power of computations in the category $DB$, are the categorial computational properties, as known, based on monads:
Proposition 4  The power-view closure 2-endofunctor $T = (T^0, T^1) : DB \to DB$ defines the monad $(T, \eta, \mu)$ and the comonad $(T, \eta^C, \mu^C)$ in $DB$, such that $\eta : I_{DB} \to T$ and $\eta^C : T \cong I_{DB}$ are natural isomorphisms, while $\mu : TT \to T$ and $\mu^C : T \to TT$ are equal to the natural identity transformation $id_T : T \to T$ (because $T \cong TT$).

Proof: It is easy to verify that all commutative diagrams of the monad $(\mu_A \circ \mu_T A = \mu_A \circ T \mu_A : \mu_A \circ T \mu_A = T \mu_A, \mu_A \circ T \eta_A = A hovered arrow)$ and the comonad are diagrams composed by identity arrows. Notice that by duality we obtain $\eta_{TA} = T \eta_A = \mu_A^{inv}$.

3.3 Duality

The following duality theorem tells us that, for any commutative diagram in $DB$, there is the same commutative diagram composed by equal objects and by inverted equivalent arrows as well. This “bidirectional” mappings property of $DB$ is a consequence of the fact that a composition of arrows is semantically based on the set-intersection commutativity property for “information fluxes” of its arrows. Thus any limit diagram in $DB$ also has its “reversed” equivalent colimit diagram with equal objects, and any universal property also has its equivalent couniversal property in $DB$.

Theorem 2 there exists the controvariant functor $S = (S^0, S^1) : DB \to DB$ such that

1. $S^0$ is an identity function on objects.
2. for any arrow in $DB$, $f : A \to B$ we have $S^1(f) : B \to A$, such that $S^1(f) = f^{inv}$ where $f^{inv}$ is an (equivalent) reversed morphism of $f$ (i.e., $f^{inv} = \tilde{f}$), $f^{inv} = i_A \circ (T f)^{inv} \circ i_B$ with $\bigcup \{q_{TB} : TB \to TA\}$.

3. The category $DB$ is equal to its dual category $DB^{OP}$.

Proof: We have, from the definition of reversed arrow, that $\widetilde{f}^{inv} = i_A \bigcap (\tilde{T f})^{inv} \bigcap i_B = T A \bigcap (\tilde{T f})^{inv} \bigcap T B = TA \bigcap \tilde{T f} \bigcap T B = \tilde{f}$. The reversed arrow of any identity arrow is equal to it, and, also, the compositional property for functor holds (the intersection operator for “information fluxes” is commutative). Thus, the controvariant functor is well defined.

It is convenient to represent this controvariant functor as a covariant functor $S : DB^{OP} \to DB$, or a covariant functor $S^{OP} : DB \to DB^{OP}$. It is easy to verify that for compositions of these covariant functors hold, $SS^{OP} = I_{DB}$ and $S^{OP}S = I_{DB^{OP}}$ w.r.t. the adjunction $< S, S^{OP}, \delta : DB^{OP} \to DB$, where $\delta$ is a bijection: for each pair of objects $A, B$ in $DB$ we have the bijection of hom-sets, $\phi_{A,B} : DB(A, S(B)) \cong DB^{OP}(S^{OP}(A), B)$, i.e., $\phi_{A,B} : DB(A, B) \cong DB(B, A)$, such that for any arrow $f \in DB(A, B)$ holds $\phi_{A,B}(f) = S^1(f) = f^{inv}$. The unit and counit of this adjunction are the identity natural transformations, $\eta_{OP} : I_{DB} \to SS^{OP}$, $\epsilon_{OP} : S^{OP}S \to$
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 $\partial_0(f + g) \equiv \partial_0(f) + \partial_0(g)$, $\partial_1(f + g) \equiv \partial_1(f) + \partial_1(g)$. 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 5** 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 $\tilde{f} + \tilde{g} = \tilde{f} + \tilde{g}$, which is a closed object, that is, holds that $T(\tilde{f} + \tilde{g}) = T\tilde{f} + T\tilde{g} = \tilde{f} + \tilde{g}$. From $T(A + A) = TA + TA \neq TA$ we obtain that $A + A$ is not isomorphic to $A$.

**Definition 8.** 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.

**Proposition 6** There exists an idempotent coproduct bifunctor $+: DB \times DB \rightarrow DB$ which is a disjoint union operator for objects and arrows in $DB$. The category $DB$ is cocartesian with initial (zero) object $\bot^0$ 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 $\bot^0$. For each pair of objects $A,B$ there exists a categorial product $A \times B$ with epimorphisms (projections) $p_A = inv_{A^{rev}}^A : A \times A \rightarrow A$ and $p_B = inv_{B^{rev}}^B : B \times B \rightarrow B$, 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 $DB \times DB$, where $id_A$, $id_B$ are the identity arrows of $A$ and $B$ respectively, holds that $id_A + id_B = \tilde{id}_A + \tilde{id}_B = TA + TB = T(A + B) = T\tilde{id}_{A + B}$. 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 $k : A \rightarrow A_1$, $k_1 : A_1 \rightarrow A_2$, $l : B \rightarrow B_1$, $l_1 : B_1 \rightarrow B_2$, holds $+^1(k_1, l_1) \circ +^1(k, l) = +^1(k_1, l_1) \cap +^1(k, l) = k_1 \circ k + l_1 \circ l = +^1(k_1 \circ k, l_1 \circ l)$. 

$\square$
For any given pair of arrows with the same codomain, there exists a unique arrow \( h \circ g \). Consequently, \( T \) has a left and a right adjunction for the diagonal functor \( \Delta : DB \to DB^2 \).

To explain these concepts in another way, we can see the limits and colimits as (with \( \tilde{\Delta} \)). Let us demonstrate the coproduct property of this bifunctor: for any two arrows \( f : A \to C \) and \( g : B \to C \), there is a unique arrow \( k : A + B \to C \), such that \( f = k \circ \text{in}_A \) and \( g = k \circ \text{in}_B \), where \( \text{in}_A : A \to A + B \) and \( \text{in}_B : B \to A + B \) are the injection (point to point) monomorphisms. The proof is by induction.

3. Let us show that for any pair of arrows \( f : A \to C \) and \( g : B \to C \), there is exactly one arrow \( k = e_C \circ (f + g) : A + B \to C \), where \( e_C : C + C \to C \) is an epimorphism (with \( e_C = TC \)), such that \( k = \tilde{f} + \tilde{g} \).

The following proposition introduces the pullbacks (and pushouts, by duality) for the category \( DB \).

**Proposition 7** For any given pair of arrows with the same codomain, \( f : A \to C \) and \( g : B \to C \), there is a pullback with the fibred product \( D = \tilde{f} \cap \tilde{g} \) (product of \( A \) and \( B \) over \( C \)). By duality, for any pair of arrows with the same domain there is a pushout as well.

**Proof:** We define the commutative diagram \( f \circ h_A = h_B \circ g \), where \( h_A : D \to A \) and \( h_B : D \to B \) are monomorphisms defined by \( h_A = i_{D}^{-1} \circ i_{DA}^{-1} \circ \text{in}_A \), \( h_B = i_{D}^{-1} \circ i_{DB}^{-1} \circ \text{in}_B \), where \( i_{DA} : D \to A \), \( i_{DB} : D \to B \) are isomorphisms and \( \text{in}_D : D \to TA \), \( \text{in}_D : D \to TB \) are monomorphisms, such that \( h_A = h_B = \tilde{f} \cap \tilde{g} = D \).

Let us show that for any pair of arrows \( l_A : E \to A \), \( l_B : E \to B \), such that \( f \circ l_A = l_B \circ g \) there is a unique arrow \( k : E \to D \) such that a pullback diagram

\[
\begin{array}{ccc}
E & \xrightarrow{k} & D \\
\downarrow{l_A} & & \downarrow{h_A} \\
A & \xrightarrow{f} & C \\
\downarrow{h_B} & & \downarrow{g} \\
B & \xrightarrow{l_B} & D \\
\end{array}
\]

commutes, i.e., (a) that \( l_A = h_A \circ k \) and \( l_B = h_B \circ k \). In fact, it must hold \( \tilde{k} \subseteq TD = T(\tilde{f} \cap \tilde{g}) = \tilde{f} \cap \tilde{g} = h_A = h_B \). So, from the commutativity (a), \( \tilde{l_A} = \tilde{h}_A \cap \tilde{k} = \tilde{k} \) and \( \tilde{l_B} = \tilde{h}_B \cap \tilde{k} = \tilde{k} \). Thus, for any other arrow \( k_1 : E \to D \) that makes a commutativity (a) must hold that \( \tilde{k_1} = \tilde{l_A} = \tilde{l_B} \) and, consequently, \( \tilde{k_1} = \tilde{k} \), i.e., \( k_1 = k \).

Consequently, \( DB \) is a cartesian category with a terminal object and pullbacks, thus it is complete (has all limits). By duality we deduce that it is also cocomplete (has all colimits).

In order to explain these concepts in another way, we can see the limits and colimits as a left and a right adjunction for the diagonal functor \( \Delta : DB \to DB^2 \) for any small
For any colimit functor $F : DB^J \to DB$ we have a left adjunction to diagonal functor $\langle F, \eta, \epsilon \rangle : DB^J \to DB$, with the colimit object $F(D)$ for any object (diagram) $D \in DB^J$ and the universal cone, a natural transformation, $\eta_C : Id_{DB^J} \to \Delta F$. Then, by duality, the same functor $F$ is also a right adjoint to the diagonal functor (adjunction, $\langle \Delta, F, \eta, \epsilon \rangle : DB \to DB^J$), with the limit object (equal to the colimit object above) $F(D)$ and the universal cone (counit), a natural transformation, $\epsilon : \Delta F \to Id_{DB^J}$, such that $\epsilon = \eta_C^{inv}$ and $\eta = \epsilon^{inv}$.

Let us see, for example, the coproducts ($F = +$) and products ($F = \times \equiv +$). In that case the diagram $D \in DB^J$ is just a diagram of two arrows with the same codomain. We obtain for the universal cone unit $\eta_C(< A, B >) : < A, B > \to < A + B, A + B >$ one pair of coproduct inclusion-monomorphisms $\eta_C(< A, B >) =< in_A, in_B >$, where $in_A : A \leftarrow A + B, in_B : B \leftarrow A + B$. The universal cone counit of product $\epsilon(< A \times B, A \times B >) : < A \times B, A \times B > \to < A, B >$ is a pair of product projection-epimorphisms $\epsilon(< A \times B, A \times B >) =< p_A, p_B >$, where $p_A : A \times B \to A, p_B : A \times B \to B, A \times B = A + B, p_A = in_{A inv}, p_B = in_{B inv}$, as represented in the following diagram:

![Diagram](image)

**Example 3:** Let us verify that each object in $DB$ is a limit of some equalizer and a colimit of its dual coequalizer. In fact, for any object $A$, a “structure map” $h : TA \to A$ of a monadic $T$-algebra $< A, h >$ derived from a monad $(T, \eta, \mu)$ (where $h \circ \eta_A = id_A$, so that $h$ is an isomorphism $h = \eta_A^{inv} = \eta_A^C$, i.e., $h = TA = \overline{id_A}$) we obtain the absolute coequalizer (by Back’s theorem, it is preserved by the endofunctor $T$, i.e., $T$ creates a coequalizer) with a colimit $A$, and, by duality, we obtain the absolute equalizer with the limit $A$ as well.

![Diagram](image)
4 Equivalence relations for databases

We can introduce a number of different equivalence relations for instance-databases:

- **Identity relation:** Two instance-databases (sets of relations) $A$ and $B$ are identical when holds the set identity $A = B$.
- **behavioral equivalence** relation: Two instance-databases $A$ and $B$ are behaviorally equivalent when each view obtained from a database $A$ can also be obtained from a database $B$ and vice versa.
- **weak observational equivalence** relation: Two instance-databases $A$ and $B$ are weakly equivalent when each "certain" view (without Skolem constants) obtained from a database $A$ can be also obtained from a database $B$ and vice versa.

It is also possible to define other kinds of equivalences for databases. In the rest of this chapter we will consider only the second and third equivalences defined above.

4.1 The (strong) behavioral equivalence for databases

Let us now consider the problem of how to define equivalent (categorically isomorphic) objects (database instances) from a behavioral point of view based on observations: as we see, 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 can characterize each object in DB (a database instance) by its behavior according to a given set of observations. Indeed, if one object $A$ is considered as a black-box, the object $TA$ is only the set of all observations on $A$. So, given two objects $A$ and $B$, we are able to define the relation of equivalence between them based on the notion of the bisimulation relation. If the observations (resulting views of queries) of $A$ and $B$ are always equal, independent of their particular internal structure, then they look equivalent to an observer.

In fact, any database can be seen as a system with a number of internal states that can be observed by using query operators (i.e, programs without side-effects). Thus, databases $A$ and $B$ are equivalent (bisimilar) if they have the same set of observations, i.e. when $TA$ is equal to $TB$:

**Definition 9.** The relation of (strong) behavioral equivalence $\approx'$ between objects (databases) in DB is defined by

$$A \approx B \text{ iff } TA = TB$$

the equivalence relation for morphisms is given by,

$$f \approx g \text{ iff } \tilde{f} = \tilde{g}$$

This relation of behavioral equivalence between objects corresponds to the notion of isomorphism in the category DB (see Proposition 2).

This introduced equivalence relation for arrows $\approx$, may be given by an (interpretation) function $B_T : \text{Mor}_{DB} \rightarrow \text{Ob}_{DB}$ (see Definition 7), such that $\approx$ is equal to the kernel of $B_T$,($\approx = \ker B_T$), i.e., this is a fundamental concept for categorial symmetry [26]:

20
Definition 10. **CATEGORICAL SYMMETRY:**

Let $C$ be a category with an equivalence relation $\cong \subseteq \text{Mor}_C \times \text{Mor}_C$ for its arrows (equivalence relation for objects is the isomorphism $\cong \subseteq \text{Ob}_C \times \text{Ob}_C$) such that there exists a bijection between equivalence classes of $\cong$ and $\simeq$, so that it is possible to define a skeletal category $[C]$ whose objects are defined by the image of a function $B_T : \text{Mor}_C \rightarrow \text{Ob}_C$ with the kernel $\ker B_T = \cong$, and to define an associative composition operator for objects $\ast$, for any fitted pair $g \circ f$ of arrows, by $B_T(g) \ast B_T(f) = B_T(g \circ f)$.

For any arrow in $C$, $f : A \rightarrow B$, the object $B_T(f)$ in $C$, denoted by $\tilde{f}$, is denominated as a conceptualized object.

**Remark:** This symmetry property allows us to consider all the properties of an arrow (up to the equivalence) as properties of objects and their composition as well. Notice that any two arrows are equal if and only if they are equivalent and have the same source and the target objects.

We have that in symmetric categories holds that $f \cong g$ iff $\tilde{f} \cong \tilde{g}$.

Let us introduce, for a category $C$ and its arrow category $C \downarrow C$, an encapsulation operator $J : \text{Mor}_C \rightarrow \text{Ob}_{C/}C$, that is, a one-to-one function such that for any arrow $f : A \rightarrow B$, $J(f) = \langle A, B, f \rangle$ is its correspondent object in $C \downarrow C$, with its inverse $\psi$ such that $\psi(\langle A, B, f \rangle) = f$.

We denote by $F_{st}, S_{nd} : (C \downarrow C) \rightarrow C$ the first and the second comma functorial projections (for any functor $F : C \rightarrow D$ between categories $C$ and $D$, we denote by $F^{0}$ and $F^{1}$ its object and arrow component), such that for any arrow $(k_{1}, k_{2}) : \langle A, B, f \rangle \rightarrow \langle A', B', g \rangle$ in $C \downarrow C$ (such that $k_{2} \circ f = g \circ k_{1}$ in $C$), we have that $F^{0}(\langle A, B, f \rangle) = A, F^{1}(k_{1}; k_{2}) = k_{1}$ and $S_{nd}^{0}(\langle A, B, f \rangle) = B, S_{nd}^{1}(k_{1}; k_{2}) = k_{2}$.

We denote by $\Delta : C \rightarrow (C \downarrow C)$ the diagonal functor, such that for any object $A$ in a category $C$, $\Delta^{0}(A) = \langle A, A, \text{id}_{A} \rangle$.

An important subset of symmetric categories are Conceptually Closed and Extended symmetric categories, as follows:

**Definition 11.** Conceptually closed category is a symmetric category $C$ with a functor $T_{e} = (T_{e}^{0}, T_{e}^{1}) : (C \downarrow C) \rightarrow C$ such that $T_{e}^{0} = B_{T}\psi$, i.e., $B_{T} = T_{e}^{0}J$, with a natural isomorphism $\varphi : T_{e} \circ \Delta \cong 1_{C}$, where $1_{C}$ is an identity functor for $C$.

$C$ is an extended symmetric category if holds also $\tau^{-1} \bullet \tau = \psi$, for vertical composition of natural transformations $\tau : F_{st} \rightarrow T_{e}$ and $\tau^{-1} : T_{e} \rightarrow S_{nd}$.

**Remark:** it is easy to verify that in conceptually closed categories, it holds that any arrow $f$ is equivalent to an identity arrow, that is, $f \cong \text{id}_{\tilde{f}}$.

It is easy to verify also that in extended symmetric categories the following holds:

$\tau = (T_{e}^{1}(\tau_{I}F_{st}^{0}; \psi)) \bullet (\varphi^{-1}F_{st}^{0}), \quad \tau^{-1} = (\varphi^{-1}S_{nd}^{0}) \bullet (T_{e}^{1}(\psi; \tau_{I}S_{nd}^{0})),$

where $\tau_{I} : I_{C} \rightarrow I_{C}$ is an identity natural transformation (for any object $A$ in $C$, $\tau_{I}(A) = \text{id}_{A}$).

**Example 4:** The $\text{Set}$ is an extended symmetric category: given any function $f : A \rightarrow B$, the conceptualized object of this function is the graph of this function (which is a set), $\tilde{f} = B_{T}(f) = \{(x, f(x)) \mid x \in A\}$.
The equivalence ≃ on morphisms (arrows) is defined by: two arrows \( f \) and \( g \) are equivalent, \( f \equiv g \), if they have the same graph.

The composition of objects * is defined as associative composition of binary relations (graphs), \( B_T(g \circ f) = \{(x, (g \circ f)(x)) \mid x \in A\} = \{(y, g(y)) \mid y \in B\} \circ \{(x, f(x)) \mid x \in A\} = B_T(g) \ast B_T(f) \).

Set is also conceptually closed by the functor \( T_e \), such that for any object \( J(f) = \langle A, B, f \rangle \), \( T^0_e(J(f)) = \{(x, f(x)) \mid x \in A\} \), and for any arrow \( (k_1; k_2) : \langle f_1; f_2 \rangle \rightarrow \langle g_1; g_2 \rangle \), the component \( T^1_e \) is defined by:

\[
\begin{align*}
T^1_e(k_1; k_2)(x, f(x)) &= (k_1(x), k_2(f(x))).
\end{align*}
\]

The composition of objects \( * \) is defined by: two arrows \( x \circ f \) and \( g \circ y \) of arrows, is the set intersection operator \( \cap \).

For example, \( \text{Set} \) is also an extended symmetric category, such that for any object \( J(f) = \langle A, B, f \rangle \) in \( \text{Set} \), \( T^0_e(J(f)) = \hat{f} \), while its arrow component \( T^1_e \) is defined as follows: for any arrow \( (h_1; h_2) : \langle f_1; f_2 \rangle \rightarrow \langle g_1; g_2 \rangle \) in \( \text{DB} \), such that \( g \circ h_1 = h_2 \circ f \) in \( \text{DB} \), holds

\[
T^1_e(h_1; h_2) = \bigcup_{\hat{q}_f \in \hat{f}} \{ q_\hat{f} \} = \{ \hat{q}_f \}
\]

Thus, \( B_T(g) \ast B_T(f) = \hat{g} \cap \hat{f} = \hat{g} \circ \hat{f} = B_T(g \circ f) \).

The associative composition operator for objects * defined for any pair of arrows \( g \circ f \) of arrows, is the set intersection operator \( \cap \).

Theorem 3 The category \( \text{DB} \) is an extended symmetric category, closed by the functor \( T_e = (T^0_e, T^1_e) : (C \downarrow C) \rightarrow C \), where \( T^0_e = B_T \psi \) is the object component of this functor such that for any arrow \( f \) in \( \text{DB} \), \( T^0_e(J(f)) = \hat{f} \), while its arrow component \( T^1_e \) is defined as follows: for any arrow \( (h_1; h_2) : J(f) \rightarrow J(g) \) in \( \text{DB} \), such that \( g \circ h_1 = h_2 \circ f \) in \( \text{DB} \), holds

\[
T^1_e(h_1; h_2) = \bigcup_{\hat{q}_f \in \hat{f}} \{ q_\hat{f} \} = \{ \hat{q}_f \}
\]

Thus, \( B_T(g) \ast B_T(f) = \hat{g} \cap \hat{f} = \hat{g} \circ \hat{f} = B_T(g \circ f) \).

Proof: Each object \( A \) has its identity (point-to-point) morphism \( id_A = \cup \{ q_\hat{id} \} \in \text{Set} \) and holds the associativity \( h \circ (g \circ f) = (h \circ g) \circ f \).

The identity arrow of \( \hat{f} \). For any two arrows \( (h_1; h_2) : J(f) \rightarrow J(g) \), it holds that \( T^1_e(h_1; h_2) \circ T^1_e(l_1; l_2) = T^1_e(h_1; h_2) \cap T^1_e(l_1; l_2) = T(l_2 \circ g) \cap T(h_2 \circ f) = l_2 \cap g \cap h_2 \cap f = (by l_2 \circ g = gh_1) = l_2 \cap g \cap h_2 \cap f \).
valid.

Remark: It is easy to verify (from $\tau^{-1} \bullet \tau = \psi$) that for any given morphism $f : A \rightarrow B$ in $DB$, the arrow $f_{ep} = \tau(J(f)) : A \rightarrow \tilde{f}$ is an epimorphism, and the arrow $f_{in} = \tau^{-1}(J(f)) : \tilde{f} \leftarrow B$ is a monomorphism, so that any morphism $f$ in $DB$ is a composition of an epimorphism and monomorphism $f = f_{in} \circ f_{ep}$, with the intermediate object equal to its "information flux" $\tilde{f}$, and with $f \approx f_{in} \approx f_{ep}$.

Let us prove that the equivalence relations on objects and morphisms are based on the "inclusion" Partial Order (PO) relations, which define the $DB$ as a 2-category:

**Proposition 8** The subcategory $DB_I \subseteq DB$, with $\text{Ob}_{DB_I} = \text{Ob}_{DB}$ and with monomorphic arrows only, is a Partial Order category with the PO relation of "inclusion" $A \preceq B$ defined by a monomorphism $f : A \hookrightarrow B$. The "inclusion" PO relations for objects and arrows are defined as follows:

$$A \preceq B \quad \text{iff} \quad TA \subseteq TB$$

$$f \preceq g \quad \text{iff} \quad \tilde{f} \preceq \tilde{g} \quad \text{(i.e., } \tilde{f} \subseteq \tilde{g})$$

they determine two observation equivalences, i.e.,

$$A \cong B \quad \text{(i.e., } A \approx B) \quad \text{iff} \quad A \preceq B \quad \text{and} \quad B \preceq A$$

$$f \approx g \quad \text{iff} \quad f \preceq g \quad \text{and} \quad g \preceq f$$

The power-view endofunctor $T : DB \rightarrow DB$ is a 2-endofunctor and a closure operator for this PO relation: any object $A$ such that $A = TA$ will be called a "closed object".

$DB$ is a 2-category, 1-cells are its ordinary morphisms, while 2-cells (denoted by $\sqrt{\alpha}$) are the arrows between ordinary morphisms: for any two morphisms $f, g : A \rightarrow B$, such that $f \preceq g$, a 2-cell arrow is the "inclusion" $\sqrt{\alpha} : f \rightarrow g$. Such a 2-cell arrow is represented by an ordinary arrow in $DB$, $\alpha : \tilde{f} \hookrightarrow \tilde{g}$, where $\alpha = T^1(\text{id}_A; \text{id}_B)$.

Proof: The relation $A \preceq B$ is well defined: any monomorphism $f : A \hookrightarrow B$ is a unique monomorphism (for any other monic arrow $g : A \hookrightarrow B$ must hold $\tilde{g} = TA = \tilde{f}$, thus $g = f$). Consequently, between any two given objects in $DB_I$ there can exist at maximum one arrow, so this is a PO category. The "inclusion" $A \preceq B$ is not a simple set inclusion $\subseteq$ between elements of $A$ and elements of $B$ (this is the case only for closed objects and, generally, $A \subseteq B$ implies $A \preceq B$, but not viceversa). The following properties are valid:

1. $A \preceq B$ implies $TA \preceq TB$ (i.e., $TA \subseteq TB$), from the definition of $\preceq$, if all elements of $A$ can define only one part of $B$, then the set of views of $A$ is a subset of the set of views of $B$; $T$ is a monotonic operator.
2. $A \preceq TA$, i.e., each element of $A$ is also a view of $A$.
3. $TA = T^2A$, as explained at the beginning of this paper.

Thus, $T$ is a closure operator, and an object $A$, such that $A = TA$ is a closed object. The rest of the proof comes directly from Proposition 2 and the definitions. Let us verify that the arrow component of this endofunctor is a closure operator as well.
1. \( f \preceq g \) implies \( Tf \preceq Tg \) (i.e., from \( f \preceq g \) holds that \( \tilde{f} \preceq \tilde{g} \), thus \( T\tilde{f} \preceq T\tilde{g} \), i.e. \( \tilde{Tf} \preceq \tilde{Tg} \))

2. \( f \preceq Tf \), from \( \tilde{f} \preceq \tilde{Tf} \)

3. \( Tf = T\tilde{Tf} \), in fact \( \tilde{Tf} = T\tilde{Tf} = T\tilde{Tf} = \tilde{Tf} \)

Notice that for each arrow \( f \) it holds (by closure property of \( T \) that \( f \approx Tf \), i.e., that \( \tilde{f} = T\tilde{f} = \tilde{Tf} \).

It is easy to verify that \( DB \) is a 2-category with 0-cells (its objects), 1-cells (its ordinary morphisms (mappings)) and 2-cells (arrows (“inclusions”) between mappings).

The horizontal and vertical composition of 2-cells is just the composition of PO reorderingsmorphisms (mappings) and 2-cells (arrows (“inclusions”) between mappings).

Example 5: Equivalent morphisms: for any view-map \( q_{A_i} : A \rightarrow TA \) the equivalence with another view-mapping \( q_{B_i} : B \rightarrow TB \) is obtained when they produce the same view.

Let us now see that each 2-cell may be represented by an equivalent ordinary morphism (1-cell) (from \( f \preceq g \) if \( f \tilde{f} \preceq \tilde{g} \)), and moreover, that we are able to treat the mappings between mappings directly as morphisms of the \( DB \) category.

The categorial symmetry operator \( T^0_\rho J : Mor_DB \rightarrow Ob_{DB} \) for any mapping (morphism) \( f \) in \( DB \) produces its “information flux” object \( \tilde{f} \) (i.e., the “conceptualized” database of this mapping). Consequently, we can define a “mapping between mappings” (which are 2-cells (“inclusions”)) and also all higher n-cells [27] by their direct transposition into a 1-cell morphism, but we are able to make more complex morphisms between mappings as well.

Example 6: Let us consider the two ordinary (1-cells) morphisms in \( DB, f : A \rightarrow B, g : C \rightarrow D \) such that \( f \preceq g \). We want to show that its 1-cells correspondent monomorphism \( \alpha : \tilde{f} \hookrightarrow \tilde{g} \) is a result of the symmetric closure functor \( T^0_\rho \). Let us prove that for two arrows, \( h_A = is_C \circ in_C \circ \tau(J(f)) \) and \( h_B = is_D \circ in_D \circ e_B \circ is_B \) (where \( in_C : \tilde{f} \hookrightarrow TC \) is a monomorphism (well defined, because \( f \preceq g \) implies \( f \subseteq g \subseteq TC \)), \( is_C : TC \rightarrow C \) is an isomorphism, \( is_B : B \rightarrow TB \) is an isomorphism, \( e_B : TB \rightarrow \tilde{f} \) is an epimorphism, \( in_D : \tilde{f} \hookrightarrow TD \) is a monomorphism (\( f \subseteq g \subseteq TD \), \( is_D : TD \rightarrow D \) is an isomorphism) holds that \( g \circ h_A = h_B \circ f \): we have that \( h_A = is_C \circ in_C \circ \tau(J(f)) = TC \cap T\tilde{f} \cap \tilde{f} = f \), (because \( TC \supseteq g \supseteq \tilde{f} \) and \( T\tilde{f} = \tilde{f} \)), and analogously \( h_B = T\tilde{f} = \tilde{f} \). Thus, \( g \circ h_A = h_B \circ f \) holds, and finally \( g \circ h_A = h_B \circ f \).

Thus, there exists the arrow \( (h_A; h_B) : J(f) \rightarrow J(g) \) in \( DB \downarrow DB \). Let us prove that also \( T^1_\rho (h_A; h_B) \) is a monomorphism as well, and that it holds that \( \alpha = T^1_\rho (h_A; h_B) \):
\( f \mapsto g \): in fact, by definition,

\[
T^1_v(h_A; h_B) = \bigcup \{ q_{f_i} \} = \bigcup \{ q_{f_i} \}
\]

because \( h_B \circ f = \tilde{f} \). Thus, \( T^1_v(h_A; h_B) = T\tilde{f} = \tilde{f} \) and, consequently, \( T^1_v(h_A; h_B) \) is a monomorphism.

In the particular case when \( A = C \) and \( B = D \) we obtain for the 2-cells arrow \( \sqrt{\alpha} : f \sim g \) represented by the 1-cell arrow \( \alpha = T^1_v(id_A; id_B) : \tilde{f} \hookrightarrow \tilde{g} \).

\[ \square \]

4.2 Weak observational equivalence for databases

A database instance can also have relations with tuples containing Skolem constants as well (for example, the minimal Herbrand models for Global (virtual) schema of some Data integration system [21, 28, 29]).

In what follows we consider a recursively enumerable set of all Skolem constants as marked (labeled) nulls \( SK = \{ \omega_0, \omega_1, \ldots \} \), disjoint from a domain set \( \text{dom} \) of all values for databases, and we introduce a unary predicate \( \text{Val}(\_ \_ \_) \), such that \( \text{Val}(t) \) is true iff \( t \in \text{dom} \) (so, \( \text{Val}(\omega_i) \) is false for any \( \omega_i \in SK \)).

Thus, we can define a new weak power-view operator for databases as follows:

**Definition 12.** Weak power-view operator \( T_w : \text{Ob}_{DB} \longrightarrow \text{Ob}_{DB} \) is defined as follows: for any database \( A \) in \( DB \) category it holds that:

\[ T_w(A) = \{ v \mid v \in T(A) \text{ and } \forall 1 \leq k \leq |v| \forall (t \in \pi_k(v))\text{Val}(t) \} \]

where \( |v| \) is the number of attributes of the view \( v \), and \( \pi_k \) is a k-th projection operator on relations.

We define a partial order relation \( \preceq_w \) for databases:

\[ A \preceq_w B \iff T_w(A) \subseteq T_w(B) \]

and we define a weak observational equivalence relation \( \equiv_w \) for databases:

\[ A \equiv_w B \iff A \preceq_w B \text{ and } B \preceq_w A. \]

The following properties hold for the weak partial order \( \preceq_w \), w.r.t. the partial order \( \preceq \) (we denote \( 'A \prec B' \) iff \( A \preceq B \) and not \( A \simeq B \)):

**Proposition 9** Let \( A \) and \( B \) be any two databases (objects in \( DB \) category), then:

1. \( T_w(A) \simeq A \), if \( A \) is a database without Skolem constants
2. \( A \prec B \) implies \( A \preceq_w B \)
3. \( A \simeq B \) implies \( A \equiv_w B \)
4. \( T_w(T_w(A)) = T(T_w(A)) = T_w(T(A)) = T_w(A) \subseteq T(A) \)

thus, each object \( D = T_w(A) \) is a closed object (i.e., \( D = T(D) \)) such that \( D \equiv_w A \)

5. \( T_w \) is a closure operator w.r.t. the "weak inclusion" relation \( \preceq_w \)
Proof: 1. From $T_w(A) \subseteq T(A)$ ($T_w(A) = T(A)$ only if $A$ is without Skolem constants).

2. If $A \prec B$ then $T(A) \subseteq T(B)$, thus $T_w(T(A)) \subseteq T_w(T(B))$, i.e., $A \preceq_w B$.

3. Directly from (4) and the fact that $A \simeq B$ if $A \preceq B$ and $B \preceq A$.

4. It holds from definition of the operator $T$ and $T_w: T_w(T_w(A)) = T(T_w(A))$ because $T_w(A)$ is the set of views of $A$ without Skolem constants and from (1). $T_w(T(A)) = \{ v \mid v \in T(A) \}$ and $\forall k \leq |w| \forall l \in \pi_k(v) \forall t \in Val(t) = T_w(A)$, from $T = TT$. Let us show that $T_w(T_w(A)) = T_w(A)$. For every view $v \in T_w(T_w(A))$, from $T_w(T_w(A)) = T(T_w(A)) \subseteq T(A)$, holds that $v \in TA$ and from the fact that $v$ is without Skolem constants it follows that $v \notin T(A)$. The converse is obvious.

5. We have that $A \preceq_w T_w(A)$, $A \preceq_w B$ implies $T_w(A) \preceq_w T_w(B)$, and $T_w(T_w(A)) = T_w(A)$. Thus, $T_w$ is a closure operator.

Notice that from point 4, the partial order $\preceq$ is a stronger discriminator for databases than the weak partial order $\preceq_w$, i.e., we can have two non isomorphic objects $A \prec B$ that are weakly equivalent, $A \approx_w B$ (for example when $A = T_w(B)$ and $B$ is a database with Skolem constants). Let us extend the notion of the type operator $T$ into the notion of the endofunctor of $DB$ category:

**Theorem 4** There exists the weak power-view endofunctor $T_w = (T_w^0, T_w^1): DB \rightarrow DB$ such that

1. for any object $A$, the object component $T_w^0(A)$ is equal to the type operator $T_w$.

2. for any morphism $f: A \rightarrow B$, the arrow component $T_w^1(f)$ is defined by

$$T_w^1(f) = inv_B^inc \circ T^1(f) \circ inv_A$$

where $inv_A: T_w(A) \leftarrow T(A)$ is a monomorphism (set inclusion) and $inv_B^inc: T(B) \rightarrow T_w(B)$ is an epimorphism (reversed monomorphism $inc_B$).

3. Endofunctor $T_w$ preserves the properties of arrows, i.e., if a morphism $f$ has a property $P$ (monic, epic, isomorphic), then also $T_w(f)$ has the same property: let $P_{mono}$, $P_{epi}$ and $P_{iso}$ are monomorphic, epimorphic and isomorphic properties respectively, then the following formula is true

$$\forall (f \in Mor_{DB})(P_{mono}(f) \equiv P_{mono}(T_w(f)) \land P_{epi}(f) \equiv P_{epi}(T_w(f)) \land P_{iso}(f) \equiv P_{iso}(T_w(f)))).$$

4. There exist the natural transformations, $\xi: T_w \rightarrow T$ (natural monomorphism), and $\xi^{-1}: T \rightarrow T_w$ (natural epimorphism), such that for any object $A$, $\xi(A) = inv_A$ is a monomorphism and $\xi^{-1}(A) = inc_A$ is an epimorphism such that $\xi(A) \approx T_w(A)$.

**Proof:** It is easy to verify that for any two arrows $f: A \rightarrow B$, $g: B \rightarrow C$, it holds that $T_w(g \circ f) \subseteq T(T_w(B)) = inc_B \circ inc_B^{-1}$, thus $T_w(g \circ f) = inc_B^{-1} \circ T^1(g \circ f) \circ inc_A = inc_C^{-1} \circ T^1(g) \circ T^1(f) \circ inc_A = inc_C^{-1} \circ inc_B \circ T^1(f) \circ inc_A = T_w(g) \circ T_w(f)$, such that $T_w$ is an endofunctor. The rest is easy to verify.

Like the monad $(T, \eta, \mu)$ and comonad $(T, \eta^C, \mu^C)$ of the endofunctor $T$, we can define such structures for the weak endofunctor $T_w$ as well.
Proposition 10 The weak power-view endofunctor \( T_w = (T^0_w, T^1_w) : DB \to DB \) defines the monad \((T^0_w, \eta_w, \mu_w)\) and the comonad \((T^1_w, \eta^C_w, \mu^C_w)\) in DB, such that \( \eta_w = \xi^{-1} \circ \eta : I_{DB} \to T^1_w \) is a natural epimorphism and \( \eta^C_w = \eta^C \circ \xi : T_w \to I_{DB} \) is a natural monomorphism (\( ' \circ ' \) is a vertical composition for natural transformations).

**Proof:** It is easy to verify that all commutative diagrams of the monad and the comonad are diagrams composed by identity arrows.

\[ \square \]

5 Categorial Semantics for Data Integration/Exchange

Data exchange [29] is a problem of taking data structured under a source schema and creating an instance of a target schema that reflects the source data as accurately as possible. Data integration [21] instead is a problem of combining data residing at different sources, and providing the user with a unified global schema of this data. Thus, in this framework the concepts are defined in a more abstract way than in the instance database framework represented in the "computation" \( DB \) category. Consequently, we require an interpretation mapping from the scheme into the instance level, which will be given categorically by functors.

5.1 Data Integration/Exchange Framework

We formalize a data integration system \( I \) in terms of a triple \( \langle G, S, M \rangle \), where

- \( G = (G_T, \Sigma_T) \) is the target schema, expanded by the new unary predicate \( Val(\_) \) such that \( Val(c) \) is true if \( c \in \text{dom} \), expressed in a language \( L_G \) over an alphabet \( A_G \), where \( G_T \) is the schema and \( \Sigma_T \) are its integrity constraints. The alphabet comprises a symbol for each element of \( G \) (i.e., relation if \( G \) is relational, class if \( G \) is object-oriented, etc.).
- \( S \) is the source schema, expressed in a language \( L_S \) over an alphabet \( A_S \). The alphabet \( A_S \) includes a symbol for each element of the sources. While the source integrity constraints may play an important role in deriving dependencies in \( M \), they do not play any direct role in the data integration/exchange framework and we may ignore them.
- \( M \) is the mapping between \( G \) and \( S \), constituted by a set of assertions of the forms

\[ (1) \quad q_S \leadsto q_G, \quad q_G \leadsto q_S \]

where \( q_S \) and \( q_G \) are two queries of the same arity, over the source schema \( S \) and over the target schema \( G \) respectively. Queries \( q_S \) are expressed in a query language \( L_{M,S} \) over the alphabet \( A_S \), and queries \( q_G \) are expressed in a query language \( L_{M,G} \) over the alphabet \( A_G \). Intuitively, an assertion \( q_S \leadsto q_G \) specifies that the concept represented by the query \( q_S \) over the sources corresponds to the concept in the target schema represented by the query \( q_G \) (similarly for an assertion of type \( q_G \leadsto q_S \)).
– Queries \( q_C(x) \), where \( x = x_1, \ldots, x_k \) is a non empty set of variables, over the global schema are conjunctive queries. We will use, for every original query \( q_C(x) \), only a lifted query over the global schema, denoted by \( q \), such that \( q := q_C(x) \land Val(x_1) \land \ldots \land Val(x_k) \).

In order to define the semantics of a data integration system, we start from the data at the sources, and specify which are the data that satisfy the global schema. A source database \( D \) for \( I = \langle G, S, M \rangle \) is constituted by one relation \( r_D \) for each source \( r \) in \( S \) (sources that are not relational may be suitably presented in the relational form by wrapper’s programs). We call global database for \( I \), or simply database for \( I \), any database for \( G \). A database \( B \) for \( I \) is said to be legal with respect to \( D \) if:

– \( B \) satisfies the integrity constraints of \( G \);
– \( B \) satisfies \( M \) with respect to \( D \).
– We restrict our attention to sound views only, which are typically considered the most natural ones in a data integration setting [21, 30].

In order to obtain an answer to a lifted query \( q \) from a data integration system, a tuple of constants is considered an answer to this query only if it is a certain answer, i.e., it satisfies the query in every legal global database.

We may try to infer all the legal databases for \( I \) and compute the tuples that satisfy the lifted query \( q \) in all such legal databases. However, the difficulty here is that, in general, there is an infinite number of legal databases. Fortunately we can define another universal (canonical) database \( can(I, D) \), that has the interesting property of faithfully representing all legal databases. The construction of the canonical database is similar to the construction of the restricted chase of a database described in [31].

**Example 7:** Let us consider the following Global-and-Local-As-View (GLAV) case when each dependency in \( M \) will be a tuple-generating dependency (tdg) of the form

\[
(2) \quad \forall x (\exists y q_S(x, y) \implies \exists z q_G(x, z))
\]

where the formula \( q_S(x) \) is a conjunction of atomic formulas over \( S \) and \( q_G(x, z) \) is a conjunction of atomic formulas over \( G \). Moreover, each target dependency in \( \Sigma_T \) will be either a tuple-generating dependency (tdg) of the form

\[
(3) \quad \forall x (\exists y \phi_G(x, y) \implies \exists z \psi_G(x, z))
\]

(we will 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 \( y_i \in y \) appears at most once in the left side),

or an equality-generating dependency (egd):

\[
(4) \quad \forall x (\phi_G(x) \implies (x_1 = x_2))
\]

where the formulae \( \phi_G(x) \) and \( \psi_G(x, y) \) are conjunctions of atomic formulae over \( G \), and \( x_1, x_2 \) are among the variables in \( x \).

Notice that this example includes as special cases both LAV (when each assertion is of the form \( q_S(x) = s(x) \), for some relation \( s \) in \( S \) and \( q_S \sim q_G \)) and GAV (when each assertion is of the form \( q_G(x, z) = g(x, z) \), for some relation \( g \) in \( G \) and \( q_G \sim q_S \)) data integration mapping in which the views are sound.
5.2 A categorial semantics of database integrity constraints

It is natural for a database schema \((A, \Sigma_A)\), where \(A\) is a schema and \(\Sigma_A\) are the database integrity constraints, to take \(\Sigma_A\) to be a tuple-generating dependency (tgD) and equality-generating dependency (egD). These two classes of dependencies together comprise the embedded implication dependencies (EID) [32] which seem to include essentially all of the naturally-occurring constraints on relational databases.

Let \((A, \Sigma_A)\) be a database schema expressed in a language \(L_D\) over an alphabet \(A_D\), where \(A\) is a schema and \(\Sigma_A = \Sigma_A^{tgD} \cup \Sigma_A^{egD}\) are the database integrity constraints (set of EIDs).

We can represent it by a schema mapping \(\Sigma_A : A \rightarrow A\), and its notation in \(DB\) can be given by an arrow, as follows:

**Proposition 11** If for a database schema \((A, \Sigma_A)\) there exists a model (instance-database) \(A\) that satisfies all integrity constraints \(\Sigma_A = \Sigma_A^{tgD} \cup \Sigma_A^{egD}\), then there exists an interpretation \(\alpha\) and its extension, a functor \(\alpha^* : \text{Sch}(A, \Sigma_A) \rightarrow DB\), where \(\text{Sch}(A, \Sigma_A)\) is the category derived from the graph (arrow) \(\Sigma_A : A \rightarrow A\) (composed by the single node \(A\), the arrow \(\Sigma_A\) and the identity arrow \(\text{id}_A : A \rightarrow A\) equal to an empty set of integrity constraints; composition of arrows in this category corresponds to the union operator), such that:

\[- \alpha^*(A) \triangleq A, \quad \text{set of relations } \alpha(R) \text{ for each predicate symbol } R, \text{ in a schema } A\]

\[- \alpha^*(\text{id}_A) \triangleq \text{id}_A : A \rightarrow A, \quad \text{identity arrow in } DB \text{ of the object } A\]

\[- \alpha^*(\Sigma_A) \triangleq (f_{tgD} \cup f_{egD}) : A \rightarrow A, \text{ where:}\]

Let \(R_{i_1}\) be the set of predicate letters used in a query \(q_{A_1}(x)\) where \(\|q_{A_1}(x)\|\) is its obtained view, and \(q_i \in O(R_{i_1}, r_i)\) be mapped into a view computation \(\alpha(q_i)\) with \(\alpha(\partial_1(q_i)) = \alpha(r_i) = \|q_{A_1}(x)\|\), then

1. for each \(i\)-th \(\text{tgD} \) \(q_{A_1}(x) \implies \exists y \ q_{A_2}(x, y) \) in \(\Sigma_A^{tgD}\), we introduce a new predicate symbol \(r_i\) with the interpretation \(\alpha(r_i) = \|q_{A_2}(x, y)\|\) (the view of \(A\) obtained from a query \(q_{A_2}(x, y)\)), and

\[ f_{tgD} \triangleq (f_{tgD}^A) \circ (\bigcup_{v_i = r_i}) \circ (\alpha(\partial_{1}(v_i)) = \alpha(r_i)) : A \rightarrow A \]

where \(\alpha(v_i)\) is an inclusion-case tuple-mapping function in 5.

2. for each \(i\)-th \(\text{egD}\) \(q_{A_1}(x) \implies (x_1 = x_2)\) in \(\Sigma_A^{egD}\), we introduce a new predicate symbol \(r_i\) with the interpretation \(\alpha(r_i) = \|q_{A_1}(x)\|\) and

\[ f_{egD} \triangleq (f_{egD}^A) \circ (\bigcup_{v_i = q_{Y_i}}) \circ (\alpha(\partial_r v_i) = \text{false}) \circ (\alpha(q_{Y_i}) = \text{false}) : A \rightarrow A \]

where \(q_{Y_i} : TA \rightarrow TA\) is a \(\text{Yes/No}\) arrow in \(DB\), and \(\alpha(\partial_{r}(v_i)) : A \rightarrow TA\) a \(\text{view-map arrow in DB}\).

\(i s_A : A \simeq TA\) is an isomorphism in \(DB\) category, and \(i s_A^{-1}\) its inverse arrow.

**Proof:** It is easy to verify that if \(\alpha^*\) satisfies the conditions in points 1 and 2, then all constraints in \(\Sigma_A\) are satisfied, so that this functor is a Lawvere’s model of \(A\). Notice that for a \(\text{Yes/No}\) arrow in \(DB\) category \(q_{Y_i} : TA \rightarrow TA\), the \(\partial_{r}(q_{Y_i}) = \perp^0\) means that for a view \(\alpha(r_i) = \|q_{A_1}(x)\|\) holds \((x_1 = x_2)\), i.e., the answer of the query \(q_{A_1}(x) \implies (x_1 = x_2)\) is \text{Yes}, and \(f_{egD} = \perp^0\), for each \(\text{egD}\) constraint in \(\Sigma_A^{egD}\).
5.3 GLAV Categorial semantics

Let us consider the most general case of GLAV mapping:

**Definition 13.** For a general GLAV data integration/exchange system \( I = \langle B, A, M \rangle \), when each tgd maps a view of one database into a view of another database, we define the following two schema mappings. \( f_A : A \rightarrow C \), \( f_B : B \rightarrow C \), where \( C \) is a new logical schema composed by a new predicate symbol \( r_i(x) \) for a formulae \( q_B(x, z) \), for every i-th tgd \( \forall x \exists y q_A(x, y) \Rightarrow \exists z q_B(x, z)\) in \( M \):

\[
\begin{align*}
 f_A & \triangleq \bigcup_{\partial_0(q_i) = R_{1i} \land \partial_1(q_i) = \partial_0(v_i) \land \partial_1(v_i) = \{ r_i \}} \{ v_i : A \rightarrow C \} \\
 f_B & \triangleq \bigcup_{\partial_0(q_{Bi}) = R_{2i} \land \partial_1(q_{Bi}) = \{ r_i \}} \{ q_{Bi} : B \rightarrow C \}
\end{align*}
\]

\( (R_{1i}, R_{2i}) \) are, respectively, the set of predicate symbols used in the query \( q_A(x, y) \) and the set of predicate letters used in the query \( q_B(x, z) \).

Note: in the particular cases (GA V and LA V), when a view of one database is mapped into the global schema; for LA V it is the opposite.

We can generalize this framework into a complex data integration/exchange system \( \langle B_k, A_k, M_k \rangle, k \in N \). Let \( \text{Sch}(I) \) be the category generated by the sketch (enriched graph) \( I \). We can now define a mapping functor from the scheme-level category into the instance level category \( DB \):

**Theorem 5** If for each \( \langle B_k, A_k, M_k \rangle, \) of the data integration/exchange system \( I = \langle B_k, A_k, M_k \rangle, k \in N \), for a given instance \( A \) of the schema \( A \) there exists the universal (canonical) instance \( B = \text{can}(I, D) \) of the global schema \( B \) legal w.r.t. \( A \), then there exists the interpretation R-algebra \( \alpha^* \) and its extension, the functor (categorial Lawvere’s model) \( \alpha^* : \text{Sch}(I) \rightarrow DB \), defined as follows:

For every single data integration/exchange system \( \langle B, A, M \rangle \):

1. for any schema arrow \( f_B : B \rightarrow C \) in \( \text{Sch}(I) \) it holds that \( B = \alpha^*(B) \triangleq \text{can}(I, D) \), and \( C = \alpha^*(C) \) is the database instance of the schema \( C \) composed by:

\[
\alpha^*(f_B) \triangleq \bigcup_{\partial_0(q_{Bi}) = R_{2i} \land \partial_1(q_{Bi}) = \{ r_i \}} \{ \alpha(q_{Bi}) : B \rightarrow C \}
\]

\( (R_{1i}, R_{2i}) \) are, respectively, the set of predicate letters used in the query \( q_A(x, y) \) and the set of predicate letters used in the query \( q_B(x, z) \);
and for any schema arrow \( f_A : A \rightarrow C \) in \( \text{Sch}(I) \), it holds: \( A = \alpha^*(A) \) is a given instance of the source schema \( A \), and

\[
\alpha^*(f_A) \triangleq \bigcup_{\partial_0(q_i) = R_{i_1}, \& \partial_1(v_i) = \{r_i\}} \{\alpha(v_i \cdot q_i) : A \rightarrow C\}
\]

where \( \alpha(v_i) : \alpha(\partial_1(q_i)) \rightarrow \alpha(r_i) \) (with \( \alpha(\partial_1(q_i)) = \pi_X(\|q_A(x)\|) \) is the projection on \( x \) of the view obtained from the query \( q_A(x, y) \)) is a function:

- inclusion case, if \( i \)-th tgd has the same direction of its implication symbol (w.r.t arrow \( f_A \))
- inverse-inclusion case, if \( i \)-th tgd has the opposite direction of its implication symbol
- equal case, if \( i \)-th tgd is an equivalence dependency relation.

2. Let \( f_{inv_A} : C \rightarrow A \) be the equivalent reverse arrow of \( \alpha^*(f_A) \) and \( f_{inv_B} : C \rightarrow B \) be the equivalent reverse arrow of \( \alpha^*(f_B) \), then, for each system \( \langle B, A, M \rangle \) we obtain the equivalent direct mapping morphisms \( f = f_{inv_B} \circ \alpha^*(f_A) : A \rightarrow B \) and \( f_{inv} = f_{inv_A} \circ \alpha^*(f_B) : B \rightarrow A \) in \( \text{DB category} \).

**Proof:** Directly from the mapping properties of \( \text{DB} \) morphisms and from the equivalent reversibility of its morphisms: each morphism in \( \text{DB} \) represents a denotational semantics for a well defined exchange problem between two database instances, so we can define a functor for such an exchange problem. Such a functor, between the schema integration level (theory) and the instance level (which is a model of this theory) is just an extended interpretation function of a particular model of R-algebra. \( \square \)

**Remark:** A solution for a data integration/exchange system does not exist always (if there exists a failing finite chase, see [28, 29] for more information), but if it exists then it is a canonical universal solution and in that case there also exists a mapping functor of the theorem above. So, this theorem can be abbreviated by: ” given a data exchange problem graph \( I = \langle B_k, A_k, M_k \rangle, k \in \mathbb{N} \), then:

\[ \exists \alpha^* : \text{Sch}(I) \rightarrow \text{DB} \text{ iff there exists a universal (canonical) solution for a corresponding data integration/exchange problem”}.

The theorem above shows how GLAV mapping can be equivalently represented by LAV and GAV mappings and shows that the query answering under IC’s can be done in the same way in LAV and GAV systems.

### 5.4 Query rewriting in GAV with (foreign) key constraints

The characteristics of the components of a data integration system in this approach [28] are as follows:

- The *global schema*, expanded by the new unary predicate \( V al(\_ ) \) such that \( V al(c) \) is true if \( c \in \text{dom} \), is expressed in the relational model with \( \Sigma_T \) (key and foreign key constraints). We assume that in such a global schema \( G \) there is exactly one key constraint for each relation.
1. **Key constraints**: given a relation $r$ in the schema, a key constraint over $r$ is expressed in the form $\text{key}(r) = \mathbf{At}$, where $\mathbf{At}$ is a set of attributes of $r$. Such a constraint is satisfied in an instance-database $A$ if for each $t_1, t_2 \in r^A$, with $t_1 \neq t_2$, we have $t_1[\mathbf{At}] \neq t_2[\mathbf{At}]$, where $t[\mathbf{At}]$ is the projection of the tuple $t$ over $\mathbf{At}$.

2. **Foreign key constraints**: a foreign key constraint is a statement of the form $r_1[\mathbf{At}] \subseteq r_2[\mathbf{Bt}]$, where $r_1, r_2$ are relations, $\mathbf{At}$ is a sequence of distinct attributes of $r_1$, and $\mathbf{Bt}$ is $\text{key}(r_2)$, i.e., the sequence $[1, \ldots, h]$ constituting the key of $r_2$. Such a constraint is satisfied in a database $A$ if for each tuple $t_1$ in $r_1^A$ there exists a tuple $t_2$ in $r_2^A$ such that $t_1[\mathbf{At}] = t_2[\mathbf{Bt}]$.

- The mapping $\mathcal{M}$ is defined following the GAV (global-as-view) approach: to each relation $r$ of the global schema $G$ we associate a query $\rho(r)$ over the source schema $S$: we assume that this query preserves the key constraint of $r$.

- For each relation $r$ of the global schema, we may compute the relation $r^D$ by evaluating the query $\rho(r)$ over the source database $D$, and compute the relation $\text{Val}$ for all constants in $\text{dom}$. The various relations so obtained define what we call the retrieved global database $\text{ret}(I, D)$. Notice that, since we assume that $\rho(r)$ has been designed so as to resolve all key conflicts regarding $r$, the retrieved global database satisfies all key constraints in $G$.

In our case, with integrity constraints and with sound mapping, the semantics of a data integration system $I$ is specified in terms of a set of legal global instance-databases, namely, those databases (they exist iff $I$ is consistent w.r.t. $D$, i.e., iff $\text{ret}(I, D)$ does not violate any key constraint in $G$) that are supersets of the retrieved global database $\text{ret}(I, D)$. 

---

Fig. 5. Functorial translation
In [28], given the retrieved global database $ret(I, D)$, we may construct inductively the canonical database $can(I, D)$ by starting from $ret(I, D)$ and repeatedly applying the following rule:

$$if \,(x_1, \ldots, x_h) \in r_{can(I,D)}[A], \,(x_1, \ldots, x_h) \notin r_{2}^{can(I,D)}[B], and the foreign key constraint $r_1[A] \subseteq r_2[B] \in \Sigma_T \subseteq G,$$ then insert in $r_{2}^{can(I,D)}$ the tuple $t$ such that

- $t[B] = (x_1, \ldots, x_h)$, and
- for each $i$ such that $1 \leq i \leq \text{arity}(r_2)$, and $i$ not in $B$, $t[i] = f_{r_2,i}(x_1, \ldots, x_h)$.

Notice that the above rule does enforce the satisfaction of the foreign key constraint $r_1[A] \subseteq r_2[B]$ by adding a suitable tuple in $r_2$: the key of the new tuple is determined by the values in $r_1[A]$, and the values of the non-key attributes are formed by means of the Skolem function symbols $f_{r_2,i}$.

Based on the results in [28], $can(I, D)$ is an appropriate database for answering queries in a data integration system. Notice that the terms involving Skolem functions are never part of certain answers. Thus, the lifted queries $q$ use the $\text{Val}(\_)$ predicate in order to eliminate the tuples with a Skolem values in $can(I, D)$.

Consequently, at the logic level, this GAV data integration system can be represented by the graph composed by two arrows (Figure 5), $\alpha : \mathcal{S} \rightarrow \Gamma$ and $\Sigma_T : \Gamma_T \rightarrow \Gamma' \,(Sch(I) \text{ denotes the category derived by this graph}).$

**Definition 14.** Functorial interpretation of this logic scheme into denotational semantic domain $DB$. $\alpha^* : Sch(I) \rightarrow DB$, is defined by two corresponding arrows (Fig. 5) $f_M : D \rightarrow ret(I, D), f_S : ret(I, D) \rightarrow can(I, D)$, where $\alpha^*(\mathcal{S}) = D$ is the extension of the source database $\mathcal{S}$, $\alpha^*(\Gamma_T) = ret(I, D)$ is the retrieved global database, $\alpha^*(\Gamma) = \alpha^*(\Gamma_T, \Sigma_T) = can(I, D)$ is the universal (canonical) instance of the global schema with the integrity constraints, and

$$f_M \triangleq \bigcup \{q_{D_i} \mid \text{where } \partial_1(q_{D_i}) \triangleq \{p^{D}(r)\}, \partial_0(q_{D_i}) \text{ is the set of all predicate symbols in the query } p(r), (p(r) \rightsquigarrow r) \in \mathcal{M}\}$$

$$f_S \triangleq \bigcup \{\alpha(v_k \cdot q_{ret_i}) \mid \partial_0(q_{ret_i}) = \partial_1(q_{ret_i}) = \{r'\}, r' \in ret(I, D), \text{where } \alpha(v_k) \text{ is an inclusion-case tuple-mapping function (in 5) for } r'\}.$$ because $ret(I, D)$ and $can(I, D)$ have the same set of predicate symbols, but the extension of each of them in $ret(I, D)$ is a subset of the extension in $can(I, D)$.

**Query rewriting coalgebra semantics:**

The naive computation is impractical, because it requires the building of a canonical database, which is generally infinite. In order to overcome this problem, a query rewriting algorithm [28] consists of two separate phases.

1. Instead of referring explicitly to the canonical database for query answering, this algorithm transforms the original lifted query $q$ into a new query $exp_G(q)$ over a global schema, called the expansion of $q$ w.r.t. $G$, such that the answer to $exp_G(q)$ over the retrieved global database is equal to the answer to $q$ over the canonical database.

2. In order to avoid the building of the retrieved global database, the query does not evaluate $exp_G(q)$ over the retrieved global database. Instead, this algorithm unfolds $exp_G(q)$ to a new query, called $\text{unf}_M(exp_G(q))$, over the source relations on
the basis of \( \mathcal{M} \), and then uses the unfolded query \( \text{unf}_\mathcal{M}(\exp_\mathcal{G}(q)) \) to access the sources.

Figure 6 shows the basic idea of this approach (taken from [28]). In order to obtain the certain answers \( q^{\mathcal{I}, D} \), the user lifted query \( q \) could in principle be evaluated (dashed arrow) over the (possibly infinite) canonical database \( \text{can}(\mathcal{I}, D) \), which is generated from the retrieved global database \( \text{ret}(\mathcal{I}, D) \). In turn, \( \text{ret}(\mathcal{I}, D) \) can be obtained from the source database \( D \) by evaluating the queries of the mapping. This query answering process instead expands the query according to the constraints in \( \mathcal{G} \), than unfolds it according to \( \mathcal{M} \), and then evaluates it on the source database.

Let us show how the symbolic diagram in Fig. 6 can be effectively represented by commutative diagrams in \( DB \), correspondent to the homomorphisms between T-coalgebras representing equivalent queries over these three instance-databases: each query in \( DB \) category is represented by an arrow, and can be composed with arrows that semantically denote mappings and integrity constraints.

**Theorem 6** Let \( \mathcal{I} = (\mathcal{G}, S, M) \) be a data integration system, \( D \) a source database for \( \mathcal{I} \), \( \text{ret}(\mathcal{I}, D) \) the retrieved global database for \( \mathcal{I} \) w.r.t. \( D \), and \( \text{can}(\mathcal{I}, D) \) the universal (canonical) database for \( \mathcal{I} \) w.r.t. \( D \).

Then, a denotational semantics for query rewriting algorithms \( \exp_\mathcal{G}(q) \) and \( \text{unf}_\mathcal{M}(q) \), for a query expansion and query unfolding respectively, are given by two (partial) functions on T-coalgebras:

\[
\begin{align*}
\text{unf}_\mathcal{M}(\_)& \triangleq T(f_M^{-1} \circ f \circ f_M) \\
\exp_\mathcal{G}(\_)& \triangleq T(f_M^{-1} \circ f \circ f_M) \\
\text{unf}_\mathcal{M}(\exp_\mathcal{G}(\_))& \triangleq T(f_M^{-1} \circ f_M) \circ f_M \\
\end{align*}
\]
where \( f_M \) and \( f_\Sigma \) are given by a functorial translation of the mapping \( M \) and integrity constraints \( \Sigma_T \).

**Proof:** Let us denote by \( q_E = \exp_G(q) \) and \( q_U = \unf_M(\exp_G(q)) \) the expanded and successively unfolded queries of the original lifted query \( q \). Then, by the query-rewriting theorem the diagrams

\[
\begin{array}{c}
TD \xrightarrow{Tf} T\text{ret}(I,D) \xrightarrow{Tf_\Sigma} T\text{can}(I,D)
\end{array}
\]

based on the composition of \( T \)-coalgebra homomorphisms \( f_M : (D, q_U) \longrightarrow (\text{ret}(I,D), q_E) \) and \( f_\Sigma : (\text{ret}(I,D), q_E) \longrightarrow \text{can}(I,D) \), commute. It is easy to verify the first two facts. Then, from the composition of these two functions, we obtain

\[
\unf_M(\exp_G(\_)) = \unf_M(\_) \exp_G(\_) = Tf_M^{\text{inv}} \circ (\exp_G(\_)) \circ f_M = Tf_M^{\text{inv}} \circ (Tf_\Sigma^{\text{inv}} \circ f_\Sigma \circ f_M)
\]

because of the duality and functorial property of \( T \).

\[
\square
\]

### 5.5 Fixpoint operator for finite canonical solution

The database instance \( \text{can}(I,D) \) can be an infinite one (see an example below), thus impossible to materialize for real applications. Thus, in this paragraph we introduce a new approach to the canonical model, closer to the data exchange approach [29]. It is not restricted to the existence of query-rewriting algorithms, and thus can be used in order to define a Coherent Closed World Assumption for data integration systems also in the absence of query-rewriting algorithms [33]. The construction of the canonical model for a global schema of the logical theory \( P_G \) for a data integration system is similar to the construction of the canonical database \( \text{can}(I,D) \) described in [28]. The difference lies in the fact that, in the construction of this revisited canonical model, denoted by \( \text{can}_M(I,D) \), for a global schema, fresh marked null values (set \( SK = \{\omega_0, \omega_1, \ldots\} \) of Skolem constants) are used instead of terms involving Skolem functions, following the idea of construction of the restricted chase of a database described in [31]. Thus, we enlarge a set of ordinary constants \( \text{dom} \) of our language by \( I_U = \text{dom} \cup SK \).

Another motivation for concentrating on canonical models is a view [34] that many logic programs are appropriately thought of as having two components, an intensional database (IDB) that represents the reasoning component, and the extensional database (EDB) that represents a collection of facts. Over the course of time, we can “apply” the same IDB to many quite different EDBs. In this context it make sense to think of the IDB as implicitly defining a transformation from an EDB to a set of derived facts: we would like the set of derived facts to be the canonical model.

Now we construct inductively the revisited canonical database model \( \text{can}_M(I,D) \) over the domain \( I_U \) by starting from \( \text{ret}(I,D) \) and repeatedly applying the following rule:
if \((x_1, \ldots, x_k) \in r_1^{can_M}(I, D)[A]\), \((x_1, \ldots, x_k) \not\in r_2^{can_M}(I, D)[B]\), and the foreign key constraint \(r_1[A] \subseteq r_2[B]\) is in \(G\), then insert in \(r_2^{can_M}(I, D)\) the tuple \(t\) such that

- \(t[B] = (x_1, \ldots, x_k)\), and
- for each \(i\) such that \(1 \leq i \leq \text{arity}(r_2)\), and \(i\) not in \(B\), \(t[i] = \omega_k\), where \(\omega_k\) is a fresh marked null value.

Note that the above rule does enforce the satisfaction of the foreign key constraint \(r_1[A] \subseteq r_2[B]\), by adding a suitable tuple in \(r_2\): the key of the new tuple is determined by the values in \(r_1[A]\), and the values of the non-key attributes are formed by means of the fresh marked values \(\omega_k\) during the application of the rule above.

The rule above defines the ”immediate consequence” monotonic operator \(T_B\) defined by:

\[
T_B(I) = I \cup \{ A \mid A \in B_G, A \leftarrow A_1 \wedge \ldots \wedge A_n \text{ is a ground instance of a rule in } \Sigma_G \text{ and } \{A_1, \ldots, A_n\} \in I \}
\]

where, at the beginning \(I = \text{rel}(I, D)\), and \(B_G\) is a Herbrand base for a global schema. Thus, \(can_M(I, D)\) is a least fixpoint of this immediate consequence operator.

**Example 8:** Suppose that we have two relations \(r\) and \(s\) in \(G\), both of arity 2 and having as key the first attribute, and that the following dependencies hold on \(G\):

\[
r_2[2] \subseteq s[1], \quad s[1] \subseteq r[1].
\]

Suppose that the retrieved global database stores a single tuple \((a, b)\) in \(r\). Then, by applying the above rule, we insert the tuple \((b, \omega_1)\) in \(s\); successively we add \((b, \omega_2)\) in \(r\), then \((\omega_2, \omega_3)\) in \(s\), and so on. Observe that the two dependencies are cyclic, and in this case the construction of the canonical database requires an infinite sequence of applications of the rules. The following table represents the computation of canonical database:

<table>
<thead>
<tr>
<th>(r^{can_M}(I, D))</th>
<th>(s^{can_M}(I, D))</th>
</tr>
</thead>
<tbody>
<tr>
<td>(a, b)</td>
<td>(b, \omega_1)</td>
</tr>
<tr>
<td>(b, \omega_2)</td>
<td>(\omega_2, \omega_3)</td>
</tr>
<tr>
<td>(\omega_2, \omega_4)</td>
<td>(\omega_4, \omega_5)</td>
</tr>
<tr>
<td>(\omega_4, \omega_6)</td>
<td>(\omega_6, \omega_7)</td>
</tr>
<tr>
<td>(\ldots)</td>
<td>(\ldots)</td>
</tr>
</tbody>
</table>

Thus, the canonical model \(can_M(I, D)\) is a legal database model for the global schema. Each certain answer of the original user query \(q(x)\), \(x = \{x_1, \ldots, x_k\}\) over a global schema is equal to the answer \(q_L(x)^{can_M}(I, D)\) of the lifted query \(q_L(x) \equiv q(x) \wedge \text{Val}(x_1) \wedge \ldots \wedge \text{Val}(x_k)\) over this canonical model. Thus, if it were possible to materialize this canonical model, the certain answers could be obtained over such a database. Often it is not possible because (as in the example above) this canonical model is infinite. In that case, we can use the revisited fixpoint semantics described in [35], based on the fact that, after some point, the new tuples added into a canonical model insert only new Skolem constants which are not useful in order to obtain certain answers (true in all models of a database). In fact, Skolem constants are not part of any certain answer to conjunctive query. Consequently, we are able to obtain a finite subset of a canonical
Let \( S \) be the \( \text{ altersor } \) \( X \) such that
\[
\text{Proposition 13}
\]
A database is large enough to obtain all certain answers.
Let us denote such a finite database by \( C_M(\mathcal{I}, \mathcal{D}) \), where
\[
r = \{(a,b), (b, \omega_2), (\omega_2, \omega_3)\}, \quad s = \{(b, \omega_1), (\omega_2, \omega_3)\}
\]
is a finite least fixpoint which can be used in order to obtain certain answers to lifted queries.

In fact, we introduced marked null values (instead of Skolem functions) in order to define and materialize such a finite database: it is not a model of the data integration system (which is infinite), but has all necessary query-answering properties: it is able to give all certain answers to conjunctive queries over a global schema. Thus it can be materialized and used for query answering, instead of query-rewriting algorithms.

The procedure for computation of a canonical database for the global schema, based on “immediate consequence” monotonic operator \( T_B \) defined in precedence, can be intuitively described as follows: it starts with an instance \( < I, \emptyset > \) which consists of \( I \), instance of the source schema, and of the empty instance \( \emptyset \) for the target (global schema). Then we chase \( < I, \emptyset > \) by applying all the dependencies in \( \Sigma_{st} \) (a finite set of source-to-target dependencies) and \( \Sigma_t \) (a finite set of target integrity dependencies) as long as they are applicable. This process may fail (if an attempt to identify two domain constants is made in order to define a homomorphism between two consecutive target instances) or it may never terminate. Let \( J_i \) and \( J_{i+1} \) denote two consecutive target instances of this process \( J_0 = \emptyset \), then we introduce a function \( C_h : \Theta \rightarrow \Theta \), where \( \Theta \) is the set of all pairs \( < I, J > \), \( I \) is a source instance and \( J \) one generated by \( I \) target instances, such that:
\[
< I, J_{i+1} > = C_h(< I, J_i >) \supseteq < I, J_i >
\]
This function is monotonic. Let us define the sets
\[
S_i = T_w(\pi_2(< I, J_i >)) = T_w(J_i)
\]
and the fixpoint operator \( \Psi : \Theta_w \rightarrow \Theta_w \), where \( \Theta_w = \{ T_w(\pi_2(S)) | S \in \Theta \} \), such that \( \Psi(T_w(\pi_2(< I, J_i >))) = T_w(\pi_2(C_h(< I, J_i >))) \), i.e., \( \Psi T_w \pi_2 = T_w \pi_2 C_h : \Theta \rightarrow \Theta_w \), and with the least fixpoint \( C_M(\mathcal{I}, \mathcal{D}) = S, S = \Psi(S) \).

**Proposition 12** [35] Let \( < I, \emptyset > \) be an initial instance that consists of \( I \), a finite instance of the source schema, and of an empty instance \( \emptyset \) for the target (global schema). Then, there exists the least fixpoint \( S \) of the function \( \Psi : \Theta_w \rightarrow \Theta_w \), which is equal to \( S = T_w \pi_2 C_h^{\omega}(< I, \emptyset >) \) for a finite \( n \).

Consequently, we can demonstrate the following algebraic property for the closure operator \( T_w \):

**Proposition 13** The closure operator \( T_w \) is algebraic, that is, given any infinite canonical database \( \text{can}(\mathcal{I}, \mathcal{D}) \), holds that
\[
T_w(\text{can}(\mathcal{I}, \mathcal{D})) = \bigcup \{ T_w(X') | X' \subseteq_{\omega} \text{can}(\mathcal{I}, \mathcal{D}) \}
\]
where \( X' \subseteq_{\omega} \text{can}(\mathcal{I}, \mathcal{D}) \) means that \( X' \) is a finite subset of \( \text{can}(\mathcal{I}, \mathcal{D}) \).

**Proof:** In fact, for \( X' = \pi_2 C_h^{\omega}(< I, \emptyset >) \) for a finite \( n \) and, consequently, finite \( X' \), such that \( X' \) is the least fixpoint of \( \Psi \), i.e., \( X' = \Psi(X') \), holds that \( T_w(\text{can}(\mathcal{I}, \mathcal{D})) = \cdots \)
Notice that each infinite canonical database of a global database schema $G$ is weakly equivalent to its finite subset (an instance-database) $C_{M}(\mathcal{I}, \mathcal{D}) = X'$, where $X' = \Psi(X')$ is a finite subset of $\text{can}(\mathcal{I}, \mathcal{D})$, that is not a model of $G$ but is obtained as the least fixpoint of the operator $\Psi$.

Thus, $\text{can}(\mathcal{I}, \mathcal{D}) \approx_{w} C_{M}(\mathcal{I}, \mathcal{D})$, where $\text{can}(\mathcal{I}, \mathcal{D})$ is an infinite model of $G$, and $C_{M}(\mathcal{I}, \mathcal{D})$ is a finite weakly equivalent object to it in $\text{DB}$ category.

## 6 Conclusion

We have presented only a fundamental overview of a new approach to the database concepts developed from an observational equivalence based on views. The main intuitive result of obtained basic database category $\text{DB}$, more appropriate than the category $\text{Set}$ used for categorial Lawvere’s theories, is to have the possibility of making synthetic representations of database mappings, and queries over databases in a graphical form, such that all mapping (and query) arrows can be composed in order to obtain the complex database mapping diagrams. Let us consider, for example, the P2P systems or mappings between databases in a complex Datawarehouse. Formally, it is possible to develop a graphic (sketch-based) tool for a meta-mapping description of complex (and partial) mappings in various contexts, with a formal mathematical background.

These, and some other, results suggest the need for further investigation of:

- The semantics for Merging and Matching database operators based on a complete database lattice, as in [36].
- The expressive power of the $\text{DB}$ category with Universal Algebra considerations.
- Monad based consideration of category $\text{DB}$ as a computation model for view-based database mappings.
- A complete investigation of all paradigms for database mappings.
- A formalization in this context of query processing in a P2P framework.

We still have not considered other important properties of this $\text{DB}$ category, such as algebraic properties for finitary representation of infinite databases, that is, locally finitely representable properties [37], or monoidal enrichments, based on concept of matching of two databases, which can be used for enriched Lawvere-s theories of sketches [38–40] in very-expressive database algebraic specification for complex inter-database mappings.

Acknowledgments: I warmly thank Eugenio Moggi and Giuseppe Rosolini for their invitation to a seminar at DISI Computer Science, University of Genova, Italy, December 2003, and for a useful discussion and correspondence that have given me the opportunity to improve an earlier version of this work.

References


39