Kleisli category and database mappings

Zoran Majkić

International Society for Research in Science and Technology,
Via Palestro 13, 00185 Roma, Italy
E-mail: majk.1234@yahoo.com

Bhanu Prasad*

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

Abstract: This paper presents the semantics of database mappings in the relational database (*DB*) category, based on the power-view monad *T* and monadic algebras. The semantics can be interpreted as a computational model of view-based mappings between databases, where each query (view-mapping) can be seen as a program, so that we can use the paradigm 'from values to computations'. The objects in this category are the database-instances. The morphisms of such *DB* category are used in order to express the semantics of view-based global and local as view (GLAV) mappings between relational databases such as those used in data integration systems. Consequently, the semantics of database mappings in this *DB* category are defined based on the power-view monad *T* and the Kleisli category for databases, which can be 'internalised' in this basic *DB* category.

Keywords: relational databases; database mappings; denotational semantics.

Reference to this paper should be made as follows: Majkić, Z. and Prasad, B. (2010) 'Kleisli category and database mappings', *Int. J. Intelligent Information and Database Systems*, Vol. 4, No. 5, pp.509–527.

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

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

1 Introduction

The notion of a monad is one of the most general mathematical notions. For instance, every algebraic theory, that is, every set of operations satisfying equational laws, can be seen as a monad. Moggi (1989, 1991) stressed the computational significance of monads and explained how they may help understand programs "as functions from values to computations". The idea of Moggi, roughly, is to give denotational semantics to computations and it presents an alternative to the conceptual gap between the intensional (operational) and the extensional (denotational) approaches to the semantics of programming languages.

The idea of monad as a model for computations, based on an endofunctor T for a given category, is that for each set of values of type A, TA is the object of computations of 'type A'. Let us explain the way we can use such denotational semantics, based on monads, in the case of relational databases. It is well-known that the relational databases are complex structures, defined by some sets of n-ary relations. In addition, the mappings between the relational databases are based on some sets of view-mappings between a source database A and a target database B. We consider the views as a universal property for databases (i.e., possible observations of the information contained in some database).

We assume that a view of a database A as the relation (set of tuples) obtained by a 'Select-Project-Join + Union' (SPJRU) query $q(\mathbf{x})$ where $\mathbf{x}$ is a list of attributes of this view. We denote by $\mathcal{L}_A$ the set of all such queries over a database A and by $\mathcal{L}_A/_{\approx}$ the quotient algebra obtained by introducing the equivalence relation $\approx$, such that $q(\mathbf{x}) \approx q'(\mathbf{x})$ if both queries result in the same relation (view). Thus, a view can be equivalently considered as a term of this quotient algebra $\mathcal{L}_A/_{\approx}$ with carrier set of relations in A and a finite arity of their operators, whose computation returns a set of tuples of this view. If this query is a finite term of this algebra then it is called a 'finitary view'. Note that a finitary view can have an infinite number of tuples also.

Example: Let a database A be a single n-ary (with $n \geq 1$ a finite integer) relation R. Then any relational query formula $q(\mathbf{x})$ of finite length over this relation R will give a single (possibly empty) relation r_q with finite number of attributes. That is, the result is the view over A obtained by this query $q(\mathbf{x})$. If we apply all possible query formulae (that are finite syntax SPJRU expressions) to this relation then we obtain the set TA of all views over this database. If the domain of values of this database is finite, thus the relation R has a finite number of tuples, then TA is a finite set of finite relations (views). If the domain is infinite and R is the n-ary Cartesian product of this domain, then R has an infinite number of tuples. In that case, any projection (i.e., simple finite query) over R will be a relation (i.e., obtained view) with a finite number of attributes but with an infinite number of tuples. In that case, the set TA is an infinite one (for example, the subset of views with only one attribute and only one tuple, for each value of this infinite domain).

DB category, a base category for the semantics of databases and mappings between them, is different from the *Set* category. It is a computational model of view-based mappings between the databases, where each query (view-mapping) can be seen as a program, so that we can use the paradigm 'from values to computations'. The objects in this category are the database-instances (a database-instance is a set of n-ary relations, i.e., a set of relational tables as in standard RDBs). The morphisms in DB category are

used in order to express the semantics of view-based global and local as view (GLAV) mappings between relational databases such as those used in data integration systems. Such morphisms in this DB category are not functions but have complex tree structures. Such an *instance level* database category DB has been introduced first time in a technical report (Majkić, 2003a) and is also used (Majkić, 2003a). General information about categories is available in classic books (Lane, 1971) while more information about this particular database category DB, with the set of its objects Ob_{DB} and set of its morphisms Mor_{DB}, is recently presented (Majkić, 2008). In this paper, we emphasise only some basic properties of this DB category, in order to make this paper more self contained.

Every object (denoted by $A,B,C,\ldots$) in this category is a database instance, composed of a set of n-ary relations $a_i \in A, i=1,2,\ldots$ that are also called 'elements of A'. The power-view operator T (as defined by Majkic, 2003a), with the domain and codomain equal to the set of all database instances, is such that for any object (database) A, the object TA denotes a database composed of the set of all views of A. The object TA, for a given database instance A, corresponds to the quotient algebra $\mathcal{L}_A/_{\approx}$, where carrier is a set of equivalence classes of closed terms of a well-defined formulae of a relational algebra, 'constructed' by Σ_R-contructors (relational operators in SPJRU algebra: select, project, join and union) and symbols (attributes of relations) of a database instance A, and constants of the attribute domains. For every object A, $A \subseteq TA$ and $TA = TTA$ are true, i.e., each (element) view of the database instance TA is also an element (view) of a database instance A.

Closed object in DB is a database A such that $A = TA$. Note that even when A is finitary (i.e., it has a finite number of relations) but having at least one relation with an infinite number of tuples then TA has an infinite number of relations (views of A) and thus TA 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.

From a *behavioural point of view that is based on observations*, we can define equivalent (categorically isomorphic) objects (database instances) as follows: each arrow (morphism) is composed of a number of 'queries' (view-maps) and each query may be seen as an *observation* over some database instance (object of DB). Thus, we can characterise each object in DB (a database instance) by its behaviour according to a given set of observations. Thus, the databases A and B are equivalent (bisimilar) if they have the same set of observable internal states, i.e., when TA is equal to TB:

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

This equivalence relation corresponds to the isomorphism of objects in DB category (Majkić, 2008). It is demonstrated that this powerview closure operator T can also be extended to arrows of DB category and hence it is an endofunctor and *defines a monad* (see Section 2).

It is already demonstrated (Majkić, 2003a; Majkić, 2008) that the basic properties of this database category DB as: its *symmetry* property (bijective correspondence between arrows and objects), and also its *duality* property (DB is equal to its dual DB^{OP}) so that each limit is a colimit as well (for example, the product is also coproduct, the pullback is also pushout, empty database $\perp^0$ is the zero object, that is, it is both initial and terminal object, etc.).

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

Majkić (2009) considered some relationships of DB and standard *set* category and introduced the categorial (functors) semantics for two basic database operations namely *matching* $\otimes$ and merging $\oplus$ such that for any two databases A and $B, A \otimes B = TA \bigcap TB$ and $A \oplus B = T(A \bigcup B)$. He also defined the algebraic database lattice and shown that DB is concrete, small and is a locally and finitely presentable (lfp) category. Moreover, he has shown that DB is V-category enriched over itself and developed a metric space and a subobject classifier for this category and demonstrated that it is a weak monoidal topos.

In this paper, we presented denotational semantics for database mappings based on the power-view endofunctor T, monadic T-(co)algebras and their computational properties in DB category, and Kleisly category of a monad T that is used for categorical semantics of database mappings and database queries. The advantage of this approach is to have denotational semantics for database mappings instead of using a standard first-order logic approach that is based on GLAV semantics (Lenzerini, 2002; Cali et al., 2003). Practical applications can be developed from this approach by having a graphical tool with a high-level interface for the mappings between databases with clear and formal denotational semantics.

The rest of the paper is organised as follows: Section 2 presents a brief introduction to DB category and its power-view monad T [based on Majkić (2003a, 2008)]. Section 3 considers universal algebra theory for databases and monadic coalgebras for database mappings. Finally, Section 4 presents the categorial semantics of database mappings, based on Kleisly category of the monad T.

In what follows we will use the special symbol □ in order to sign the end of proofs and examples.

2 Monad over DB category

In this section, we present a brief introduction to DB category, based on the existing works (Majkić, 2003a, 2008). We assume that the domain of every database is an arbitrary large and finite set by default. This assumption is reasonable for all real applications. We define a universal database instance Υ, as the union of all database instances, i.e., $\Upsilon = \left\{ a_i \middle| a_i \in A, A \in Ob_{DB} \right\}$. Υ is the top object of this category, such that $\Upsilon = T\Upsilon$, because every view $v \in T\Upsilon$ is also a database instance and thus $v \in \Upsilon$, vice versa, every element $r \in \Upsilon$ is also a view of Υ, thus $r \in T\Upsilon$.

Every object (database) A contains an empty relation $\perp$ as well. The object composed only by this empty relation is denoted by $\perp^0$ and we have that $T\perp^0 = \perp^0 = \{\perp\}$. Any empty database (i.e., a database with only empty relations) is isomorphic to this bottom object $\perp^0$.

Morphisms of this category are all possible mappings between database instances *based on views*. Elementary view-map for a given database A is given by a SPCU query $f_i = q_{A_i} : A \to TA$. Let us denote the extension of the relation obtained from this query

q_{A_i} by $\|f_i\|$. Suppose that $r_{i1},\ldots,r_{ik} \in A$ are the relations used for the computation of this query, and that the correspondent algebraic term $\widehat{q_i}$ is a function (it is not a T-coalgebra) $\widehat{q_i} : A^k \to TA$, where A^k is kth Cartesian product of A. Then, $\|q_{A_i}\| = \widehat{q_i}(r_{i1},\ldots,r_{ik})$. Unlike the algebra term $\widehat{q_i}$ which is a function, a view-map $q_{A_i} : A \to TA$ *is not* a function but a *T-coalgebra*.

Consequently, an atomic morphism $f : A \to B$, from a database A to database B, is a set of such view-mappings and generally it is not a function, making the DB category different from the Set category where morphisms are functions.

We can introduce two functions, $\partial_0, \partial_1 : Mor_{DB} \to \mathcal{P}(Y)$ (which are different from standard category functions $dom, cod : Mor_{DB} \to Ob_{DB}$), such that for any view map $q_{A_i} : A \to TA$, we have that $\partial_0(q_{A_i}) = (r_1,\ldots,r_k) \subseteq A$ is a subset of relations of A used as arguments by this query q_{A_i} and $\partial_1(q_{A_i}) = \{v\}$, where $v \in TA$ is the resulting view of a query q_{A_i}. Here $\mathcal{P}$ is a power set operation. In fact, these two functions are such that for any morphism $f : A \to B$ between databases A and B, which is a set of view-mappings q_{A_i} such that $\|q_{A_i}\| \in B$, we have that $\partial_0(f) \subseteq A$ and $\partial_1(f) \subseteq TA \cap B \subseteq B$. Thus, we have:

$$\partial_0(f) = \bigcup_{q_{A_i} \in f} \partial_0(q_{A_i}) \subseteq dom(f) = A, \quad \partial_1(f) = \bigcup_{q_{A_i} \in f} \partial_1(q_{A_i}) \subseteq cod(f) = B$$

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

Definition 1 (Majkić, 2008): The following BNF defines the set Mor_{DB} of all morphisms in DB:

p-arrow: = *c-arrow* | *c-arrow* o *c-arrow* (for any two c-arrows $f : A \to B$ and $g : B \to C$)

morphism:= *p-arrow* | *c-arrow* o *p-arrow* (for any p-arrow $f : A \to B$ and c-arrow $g : B \to C$)

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

$$\begin{aligned} h = g \circ f = & \bigcup_{q_{Bj} \in q \;\&\; \partial_0(q_{Bj}) \cap \partial_1(f) \neq \varnothing} \{q_{Bj}\} \quad \circ \\ \circ & \bigcup_{q_{Ai} \in f \;\&\; \partial_1(q_{Ai}) = \{v\} \;\&\; v \in \partial_0(q_{Bj})} \{q_{Ai}(tree)\} \\ = & \left\{ q_{Bj} \circ \left\{ q_{Ai}(tree) \middle| \partial_1(q_{Ai}) \subseteq \partial_0(q_{Bj}) \right\} \middle| q_{Bj} \in g \;\&\; \partial_0(q_{Bj}) \cap (f) \neq \emptyset \right\} \\ = & \left\{ q_{Bj}(tree) \middle| q_{Bj} \in g \;\&\; \partial_0(q_{Bj}) \cap (f) \neq \emptyset \right\}, \end{aligned}$$

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

We define the semantics of any morphism $h: A \to C$ as an 'information transmitted flux' from the source to the target object. An 'information flux' (denoted by $\tilde{h}$) is a set of views (as a result, 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 $f: A \to B$ from a database A into a database B, composed of only one view map based on a single query $q(\mathbf{x}) \leftarrow R_1(u_1),\ldots,R_n(u_n)$, where $n \geq 0$ and R_i are the names of the relations (at least one relation) that are in $\mathcal{A}$ or built-in predicates (e.g., $\leq$, $=$, etc.) and q is the name of a relation that is not in A. Then, for any tuple **c** for which the body of this query is true, $q(\mathbf{c})$ must also 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 2 (Majkić, 2008): We define the semantics of mappings by function $B_T: Mor_{DB} \to Ob_{DB}$, which, given any mapping morphism $f: A \to B$, 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 $g: A \to B$ be a morphism with a flux $\tilde{g}$, and $f: B \to C$ an atomic morphism with flux $\tilde{f}$ defined in point 1, then $\widetilde{f \circ g} = B_T(f \circ g) \triangleq \tilde{f} \bigcap \tilde{g}$.

Thus, we have the following fundamental property:

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

Proof: This proposition can 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 the intersection of closed objects is always a closed object. □

Remark: The 'information flux' $\tilde{f}$ of a given morphism (mapping) $f: A \to B$ is an instance-database as well (its elements are the views defined by the formulae above), thus, an object in DB as well.

Proposition 2 (Majkić, 2008): The following properties for morphisms are valid:

1 each arrow $f: A \twoheadrightarrow B$, such that $\tilde{f} = TB$ is an epimorphism [epic arrow (Lane, 1971)]

2 each arrow $f: A \rightarrowtail B$, such that $\tilde{f} = TA$ is a monomorphism [monic arrow (Lane, 1971)]

3 each monic and epic arrow is an isomorphism, thus two objects A and B are isomorphic iff $TA = TB$, that is, $A \simeq B$ iff $TA = TB$.

Note that in the standard *set* category, where the objects of the category are sets and the arrows are functions between them, *monic* and *epic* arrows are injective and surjective functions respectively. However, in this DB category, the arrows are not functions but complex trees of functions.

Note that between any two databases A and B there is at least an 'empty' arrow $\phi : A \to C$ such that $\partial_0(\emptyset) = \partial_1(\emptyset) = \tilde{\emptyset} = \{\bot\} = \bot^0$. We have that $\bot \in A$ for any database A (in DB all objects are pointed by $\bot$ relation), so that any arrow $f : A \to B$ has as one component the empty mapping (thus, also arrows are pointed by $\emptyset$).

If f is epic then $TA \supseteq TB$, if it is monic then $TA \subseteq TB$. Thus we have an isomorphism of two objects (databases), $A \simeq B$ iff $TA = TB$.

We define an ordering $\preceq$ between databases by $A \preceq B$ iff $TA \subseteq TB$.

Thus, for any database A we have that $A \simeq TA$, i.e., there is an isomorphic arrow $is_A = \{q_{A_i} \mid \partial_0(q_{A_i}) = \partial_1(q_{A_i}) = \{v\}$ and $v \in A\} : A \to TA$ and its inverse $is_A^{inv} = \{q_{TA_i} \mid \partial_0(q_{TA_i}) = \partial_1(q_{TA_i}) = \{v\}$ and $v \in A\} : A \subseteq TA\} : TA \to A$, such that their flux is $\widetilde{is_A} = is_A^{inv} = TA$.

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

Theorem 1 (Majkić, 2008): there exists the controvariant functor $\underline{S} = (\underline{S}^0, \underline{S}^1) : DB \to DB$ such that:

1 $\underline{S}^0$ is the identity function on objects

2 for any arrow in DB, $f : A \to B$ we have $\underline{S}^1(f) : B \to A$, such that $\underline{S}^1(f) \triangleq f^{inv}$, where f^{inv} is (equivalent) reversed morphism of f (i.e., $\widetilde{f^{inv}} = \tilde{f}$), $f^{inv} = is_A^{-1} \circ (Tf)^{inv} \circ is_B$ with

$$(Tf)^{inv} \triangleq \bigcup_{\partial_0(q_{TB_j}) = \partial_1(q_{TB_j}) = \{v\} \,\&\, v \in \tilde{f}} \{q_{TB_j} : TB \to TA\}$$

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

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

Theorem 2 (Majkić, 2008): There exists the endofunctor $T = (T^0, T^1) : DB \to DB$, such that:

1 for any object A, the object component T^0 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:

$$T(f) \triangleq T^1(f) = \bigcup_{\partial_0(q_{TA_i})=\partial_1(q_{TA_i})=\{v\}\ \&\ v\in\tilde{f}} \{q_{TA_i} : TA \to TB\}$$

3 endofunctor T preserves 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_{mono}, P_{epi} and P_{iso} are monomorphic, epimorphic and isomorphic properties respectively.

Proof: it can be found in the paper by Majkić (2008) □

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

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 corresponding monad-comprehension for the monad T, which invalidates the thesis (Walder, 1990) that 'monads ≡ monad-comprehensions'. It is only valid that 'monad-comprehension ⇒ monads'.

We have already seen that the views of some database can be seen as its observable computations: to obtain an expressive power of computations in the category DB, we need categorial computational properties, as known, based on monads.

Proposition 3: The power-view closure 2-endofunctor $T = (T^0, T^1) : DB \to DB$ defines the monad (T, η, μ) and the comonad (T, η^C, μ^C) in DB, such that $\eta : I_{DB} \simeq T$ and $\eta^C : T \simeq I_{DB}$ 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 $T = TT$).

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

3 Categorial symmetry and behavioural equivalence

Let us consider the problem of how to define equivalent (categorically isomorphic) objects (database instances) from a *behavioural point of view based on observations*. As we can 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 characterise each object in DB (a database instance) by its behaviour according to a given set of observations. Indeed, if one object A is considered as a blackbox, 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 3 (Majkić, 2008): The relation of (strong) behavioural equivalence '$\approx$' between objects (databases) in DB is defined by:

$$A \approx B \; iff \; TA = TB$$

the equivalence relation for morphisms is given by, $f \approx g \; iff \; \tilde{f} = \tilde{g}$.

This relation of behavioural 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 a (interpretation) function $B_T : Mor_{DB} \to Ob_{DB}$ (see Definition 2) such that $\approx$ is equal to the kernel of B_T, $\approx = kerB_T$), i.e., this is a fundamental concept for categorial symmetry (Majkić, 1998):

Definition 4 (Majkić, 1998), categorial symmetry: Let C be a category with an equivalence relation $\approx \subseteq Mor_C \times Mor_C$ for its arrows (equivalence relation for objects is the isomorphism $\simeq \subseteq Ob_C \times Ob_C$) such that there exists a bijection between equivalence classes of $\approx$ and $\simeq$, so that it is possible to define a skeletal category $|C|$ whose objects are defined by the imagine of a function $B_T : Mor_C \to Ob_C$ with the kernel $kerB_T = \approx$, and to define an associative composition operator for objects *, for any fitted pair $g \circ f$ of arrows, by $B_T(g) * B_T(f) = B_T(g \circ f)$.

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

In symmetric categories, $f \approx g$ iff $\tilde{f} \simeq \tilde{g}$.

Let us introduce, for a category C and its arrow category $C \downarrow C$, an encapsulation operator $J : Mor_C \to Ob_{C\downarrow C}$, that is, a one-to-one function such that for any arrow $f : A \to B, J(f) = < A, B, f >$ is its correspondent object in $C \downarrow C$, with its inverse ψ such that $\psi(< A, B, f >) = f$.

We denote the first and the second comma functorial projections by $F_{st}, S_{nd} : (C \downarrow C) \to C$ (for any functor $F : C \to D$ between categories C and D, we denote its object and arrow component by F^0 and F^1), such that for any arrow $(k_1; k_2) : < A, B, f > \to < A', B', g >$ in $C \downarrow C$ (such that $k_2 \circ f = g \circ k_1$ in C), we have that $F^0_{st}(< A, B, f >) = A, F^1_{st}(k_1; k_2) = k_1$ and $S^0_{nd}(< A, B, f >) = B, S^1_{nd}(k_1; k_2) = k_2$.

We denote the diagonal functor by $\blacktriangle : C \to (C \downarrow C)$, such that for any object A in a category C, $\blacktriangle^0(A) = < A, A, id_A >$. An important subset of symmetric categories are conceptually closed and extended symmetric categories, as follows:

Definition 5 (Majkić, 1998): Conceptually closed category is a symmetric category C with a functor $T_e = (T^0_e, T^1_e) : (C \downarrow C) \to C$ such that $T^0_e = B_T \psi$, i.e., $B_T = T^0_e J$, with a natural isomorphism $\varphi : T_e \circ \blacktriangle \simeq I_C$, where I_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 $\tau : F_{st} \to T_e$ and $\tau^{-1} : T_e \to S_{nd}$.

Remark: it is easy to verify that in conceptually closed categories, any arrow f is equivalent to an identity arrow, that is, $f \approx id_{\tilde{f}}$.

It is easy to verify also that in extended symmetric categories the following is true: $\tau(T_e^1(\tau_I F_{st}^0;\psi))\bullet(\varphi^{-1}S_{nd}^0)\bullet(T_e^1(\psi;\tau_I S_{nd}^0))$, where $\tau_I : I_C \to I_C$ is an identity natural transformation (for any object A in C, $\tau_I(A) = id_A$).

Example 1: The *set* is an extended symmetric category: given any function $f : A \to B$, the conceptualised 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 $\approx$ on morphisms (arrows) is defined as follows: two arrows f and g are equivalent, i.e., $f \approx g$, iff they have the same graph.

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

Set is also conceptually closed by the functor T_e such that for any object $J(f) =< A, B, f >, T_e^0(J(f)) = B_T(f) = \{(x, f(x)) \mid x \in A\}$ and for any arrow $(k_1;k_2) : J(f) = J(g)$, the component T_e^1 is defined as: for any $(x, f(x)) \in T_e^0(J(f))$, $T_e^1(k_1;k_2)(x, f(x)) = (k_1(x), k_2(f(x)))$.

It is easy to verify the compositional property for T_e^1 and that $T_e^1(id_A; id_B) = id_{T_e^0(J(f))}$.

For example, *set* is also an extended symmetric category, such that for any object $J(f) =< A, B, f >$ in $Set \downarrow Set$, we have that $\tau(J(f)) \mid A \twoheadrightarrow B_T(f)$ is an epimorphism, such that for any $x \in A, \tau(J(f))(x) = (x, f(x))$, while $\tau^{-1}(J(f)) : B_T(f) \hookrightarrow B$ is a monomorphism such that for any $(x, f(x)) \in B_T(f)$, $\tau^{-1}(J(f))(x, f(x)) = f(x)$.

Thus, each arrow in *Set* is a composition of an epimorphism and a monomorphism. □

Now we are ready to present a formal definition for the DB category:

Theorem 3 (Majkić, 1998): The category DB is an extended symmetric category, closed by the functor $T_e = (T_e^0, T_e^1) : (C \downarrow C) \to C$, where $T_e^0 = B_T\psi$ is the object component of this functor such that for any arrow f in DB, $T_e^0(J(f)) = \tilde{f}$, while its arrow component T_e^1 is defined as follows: for any arrow $(h_1;h_2) : J(f) \to J(g)$ in $DB \downarrow DB$, such that $g \circ h_1 = h_2 \circ f$ in DB, holds:

$$T_e^1(h_1;h_2) = \bigcup_{\partial_0\left(q_{\tilde{f}_i}\right)=\partial_1\left(q_{\tilde{f}_i}\right)=\{v\}\ \&\ v\in\widetilde{h_2\circ f}} \left\{q_{\tilde{f}_i}\right\}$$

The associative composition operator for objects *, defined for any fitted pair $g \circ f$ of arrows, is the set intersection operator $\bigcap$.

Thus, $B_T(g) * B_T(f) = \tilde{g} \bigcap \tilde{f} = \widetilde{g \circ f} = B_T(g \circ f)$.

Proof: Each object A has its identity (point-to-point) morphism $id_A = \bigcap_{\partial_0(q_{A_i})=\partial_1(q_{A_j})=\{v\}\&v\in A} \{q_{A_i}\}$ and holds the associativity $h \circ\widetilde{(h \circ g)} = \tilde{h} \bigcap \widetilde{(g \circ f)}$ $= \tilde{h} \bigcap \tilde{g} \bigcap \tilde{f} = \widetilde{(h \circ g)} \bigcap \tilde{f} = \widetilde{(h \circ g)} \circ f$. They have the same source and target object, thus $h \circ (g \circ f) = (h \circ g) \circ f$. Thus, DB is a category. It is easy to verify that also T_e is a well defined functor. In fact, for any identity arrow $(id_A; id_B) : J(f) \to J(f)$ it holds that $T_e^1(id_A; id_B) = \bigcup_{\partial_0\left(q_{\tilde{f}_i}\right)=\partial_1\left(q_{\tilde{f}_i}\right)=\{v\}\ \&\ v\in\widetilde{id_B\circ f}} \{q_{\tilde{f}_i}\} = id_{\tilde{f}}$ is the identity arrow of $\tilde{f}$. For any two arrows $(h_1;h_2) : J(f) \to J(g), (l_1;l_2) : J(g) \to J(k)$, it holds that $\widetilde{T_e^1(h_1;h_2) \circ T_e^1(l_1;l_2)}$ $T_e^1(\widetilde{h_1;h_2}) \bigcap T_e^1(l_1;l_2) = \widetilde{T(l_2 \circ g)} \bigcap \widetilde{T(h_2 \circ g)} = \widetilde{l_2} \bigcap \tilde{g} \bigcap \widetilde{h_2} \bigcap \tilde{f} = (by\ l_2 \circ f = g \circ h_1) = \widetilde{l_2} \bigcap \tilde{g} \bigcap \widetilde{h_1} \bigcap \widetilde{h_2} \bigcap \tilde{f}$ $(by\ l_2 \circ f = g \circ h_1) = \widetilde{l_2} \bigcap \widetilde{h_2} \bigcap \tilde{f} = \widetilde{l_2 \circ h_2 \circ f} = T_e^1(l_1 \circ h_1; l_2 \circ h_2)$, finally $T_e^1(h_1;h_2) \circ T_e^1(l_1;l_2)$

$= T_e^1(l_1 \circ h_1; l_2 \circ h_2)$. For any identity arrow, it holds that id_A, $T_e^0 J(id_A) = \widetilde{id_A} = TA \simeq A$ as well, thus, an isomorphism $\varphi : T_e \circ \blacktriangle \simeq I_{DB}$ is valid. □

Remark: It is easy to verify (from $\tau^{-1} \bullet \tau = \psi$) that for any given morphism $f : A \to B$ in DB, the arrow $f_{ep} = \tau(J(f)) : A \twoheadrightarrow \tilde{f}$ is an epimorphism, and the arrow $f_{in} = \tau^{-1}(J(f)) : \tilde{f} \hookrightarrow B$ 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 $f \approx f_{in} \approx f_{ep}$.

4 Database mappings and monadic coalgebras

The notion of a monad is one of the most general mathematical notions. For instance, every algebraic theory, that is, every set of operations satisfying equational laws, can be seen as a monad (which is also a *monoid* in a category of endofunctors of a given category: the 'operation' μ being the associative multiplication of this monoid and η is its unit). Thus, monoid laws of the monad do subsume all possible algebraic laws.

In order to explore universal algebra properties (Majkić, 2009a; 2009b) for the category DB, where, generally, morphisms are not functions (this fact complicates a definition of mappings from its morphisms into homomorphisms of the category of Σ_R-algebras), we will use an equivalent to DB 'functional' category, denoted by DB_{sk}, such that its arrows can be seen as total functions.

Proposition 4: Let us denote by DB_{sk} the full skeletal subcategory of DB, composed by closed objects only. Such a category is equivalent to the category DB, i.e., there exists an adjunction of a surjective functor $T_{sk} : DB \to DB_{sk}$ and an inclusion functor $In_{sk} : DB_{sk} \to DB$, where In_{sk}^0 and In_{sk}^1 are two identity functions, such that $T_{sk} In_{sk} = Id_{DB_{sk}}$ and $T_{sk} In_{sk} \simeq Id_{DB}$.

There exists the faithful forgetful functor $F_{sk} : DB_{sk} \to Set$, and $F_{DB} : F_{sk} \circ T_{sk} : DB \to Set$, thus DB_{sk} and DB are concrete categories.

Proof: It can be found in Majkić (2009a). The skeletal category DB_{sk} has closed objects only, so, for any mapping $f : A \to B$, we obtain the arrow $f_T = T_{sk}^0(A) = TA \to TB$ can be expressed in a following 'total' form such that $\partial_0(f_T) = T_{sk}^0(A) = TA$:

$$f_T \triangleq \bigcup_{\partial_0(q_{TA_i}) = \partial_1(q_{TA_i}) = \{v\} \,\&\, v \in \tilde{f}} \{q_{TA_i}\} \qquad \bigcup_{\partial_0(q_{TA_i}) = \{v\} \,\&\, v \notin \tilde{f} \,\&\, \partial_1(q_{TA_i}) = \perp^0} \{q_{TA_i}\}$$

so that $f_R = F_{sk}^1(f_T) : TA \to TB$ (the component for objects F_{sk}^1 is an identity) is *a function* in *Set*, $f_R = F_{DB}^1(f)$, such that for any $v \in TA$, $f_R = (v) = v$ if $v \in \tilde{f}; \perp$ otherwise. □

In a given inductive definition, the value of a function (in our example the endofunctor T) is defined on all (algebraic) constructors (relational operators). What follows is based on the fundamental results of the universal algebra (Cohn, 1965).

Let Σ_R be a finitary signature (in the usual algebraic sense: a *finite* collection F_Σ of function symbols together with a function $ar : F_\Sigma \to N$ giving the finite arity of each function symbol) for a single-sorted (sort of relations) relational algebra.

We can speak of Σ_R-equations and their satisfaction in a Σ_R-algebra, obtaining the notion of a (Σ_R, E)-algebra theory. In a special case, when E is empty, we obtain a purely syntax version of universal algebra, where $\mathcal{K}$ is a category of all Σ_R-algebras, and the quotient algebras are simply algebras of terms.

An *algebra* for the algebraic theory (type) (Σ_R, E) is given by a set X, called the *carrier* of the algebra, together with interpretations for each of the function symbols in Σ_R. A function symbol $f \in \Sigma_R$ of arity k must be interpreted by a function $\widehat{f_X} : X^k \to X$. Given this, a term containing n distinct variables gives rise to a function $X^n \to X$ defined by induction on the structure of the term. An algebra must also satisfy the equations given in E in the sense that equal *terms* give rise to identical functions (with obvious adjustments where the equated terms do not contain exactly the same variables). A *homomorphism* of algebras from an algebra X to an algebra Y is given by a function $g : X \to Y$ which commutes with operations of the algebra $g(\hat{f}_X(x_1,\ldots,x_k)) = \hat{f}_Y(g(x_1),\ldots,g(x_k))$. This generates a variety category $\mathcal{K}$ of all relational algebras. Consequently, there is a bifunctor $E : DB_{sk}^{OP} \times \mathcal{K} \to Set$ (where *Set* is the category of sets), such that for any database instance A in DB_{sk} there exists the functor $E(A,\_) : \mathcal{K} \to Set$ with an universal element $(U(A), \varrho)$, where $\varrho \in E(A(U), A), \varrho : A \to U(A)$ is an inclusion function and $U(A)$ is a free algebra over A (quotient algebra generated by a carrier database instance A), such that for any function $f \in E(A, X)$ there is a unique homomorphism h from the free algebra $U(A)$ into an algebra X, with $f = E(A,h) \circ \varrho$.

From the so called 'parameter theorem' (Theorem 3 for *adjunctions with a parameter* in Lane (1971) for a bifunctor $E : DB_{sk}^{OP} \times \mathcal{K} \to Set$), we obtain that there exists:

- A unique universal functor $U : DB_{sk} \to \mathcal{K}$ such that for any given database instance A in DB_{sk} it returns with the free Σ_R-algebra $U(A)$ (which is a quotient algebra), where a carrier is a set of equivalence classes of closed terms of a well-defined formulae of a relational algebra, 'constructed' by Σ_R-constructors (relational operators: select, project, join and union SPJRU) and symbols (attributes and relations) of a database instance A, and constants of attribute domains. An alternative for $U(A)$ is given by considering A as a set of variables rather than a set of constants then we can consider $U(A)$ as being a set of *derived operations* of arity A for this theory. In either case, the operations are interpreted syntactically as $\hat{f}([t_1],\ldots,[t_k]) = [f(t_1,\ldots,t_k)]$. In this, brackets denote equivalence classes, while, for any 'functional' morphism (correspondent to the total function $F_{sk}^1(f_T)$ in Set, $F_{sk} : DB_{sk} \to Set) f_T : A \to B$ DB_{sk}) we obtain the homomorphism $f_H = U^1(f_T)$ from the Σ_R-algebra $U(A)$ into the Σ_R-algebra $U(B)$, such that for any term $\rho(a_1,\ldots,a_n) \in U(A), \rho \in \Sigma_R$, we obtain $f_H(\rho(a_1,\ldots,a_n)) = \rho(f_H(a_1),\ldots,f_H(a_n))$. So, f_H is an identity function for algebraic operators and it is equal to the function $F_{sk}^1(f_T)$ for constants.
- Its adjoint forgetful functor $F : \mathcal{K} \to DB_{sk}$, such that for any free algebra $U(A)$ in $\mathcal{K}$ the object $F \circ U(A)$ in DB_{sk} is equal to its carrier-set A (each term $\rho(a_1,\ldots,a_n) \in U(A)$ is evaluated into a view of this closed object A in DB_{sk}) and for each arrow $U^1(f_T)$ holds that $F^1U^1(f_T) = f_T$, i.e., $FU = Id_{DB_{sk}}$ and $UF = Id_{\mathcal{K}}$.

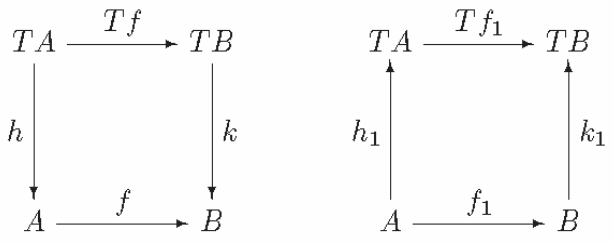
Consequently, $U(A)$ is a quotient algebra, where carrier is a set of equivalence classes of closed terms of a well-defined formulae of a relational algebra, 'constructed' by Σ_R-constructors (relational operators in SPJRU algebra: select, project, join and union)

and symbols (attributes of relations) of a database instance A and constants of attribute domains.

From the universal property, it immediately follows that the map $A \mapsto U(A)$ extends to the endofunctor $F \circ U : DB_{sk} \to DB_{sk}$. This functor carries monad structure $(F \circ U, \eta, \mu)$ with $F \circ U$ an equivalent version of T but for this skeletal database category DB_{sk}. The natural transformation η is given by the obvious 'inclusion' of A into $F \circ U(A) : a \to [a]$ (each view a in a closed object A is an equivalence class of all algebra terms which produce this view). Note that the natural transformation η is the unit of this adjunction of U and F and that it corresponds to an inclusion function in the $Set, \varrho : A \to U(A)$, given above. The interpretation of μ is almost equally simple. An element of $(F \circ U)^2(A)$ is an equivalence class of terms built up from elements of $F \circ U(A)$, so that instead of $x_1, \ldots, x_k$ a typical element of $(F \circ U)^2(A)$ is given by the equivalence class of a term $[t(t_1, \ldots, t_k)]$. The transformation μ is defined by the mapping $[t([t_1], \ldots, [t_k])] \mapsto [t(t_1, \ldots, t_k)]$. This makes sense because a substitution of provably equal expressions into the same term results in provably equal terms.

We use monads (Lane, 1971; Lambek and Scott, 1986; Kelly and Power, 1993) for giving *denotational semantics to database mappings*. More specifically, we use monads as a way of modelling computational/collection types (Moggi, 1989, 1991; Buneman et al., 1995; Plotkin and Power, 2001): to interpret database mappings (morphisms) in the category DB, we distinguish the object A (database instance of type A) from the object TA of observations (computations of type A without side-effects) and take as a denotation of (view) mappings the elements of TA (which are view of (type) A). In particular, we identify the type A with the object of values (of type A) and obtain the object of observations by applying the unary type-constructor T (power-view operator) to A. It is well-known that each endofunctor defines algebras and coalgebras (the left and right commutative diagrams):

Figure 1 Algebras and coagebras



We use the following well-known definitions in the category theory (the set of all arrows in a category $\mathcal{M}$ from A to B is denoted by $\mathcal{M}(A, B)$):

Definition 6: The categories CT_{alg} of T-algebras, CT_{coalg} of T-coalgebras, derived from an endofunctor T, are defined (Asperti and Longo, 1991) as follows:

1 the objects of CT_{alg} are pairs (A, h) with $A \in Ob_{DB}$ and $h \in DB(TA, A)$, the arrows between objects (A, h) and (B, k) are all arrows $f \in DB(A, B)$ such that $k \circ Tf = f \circ h : TA \to B$

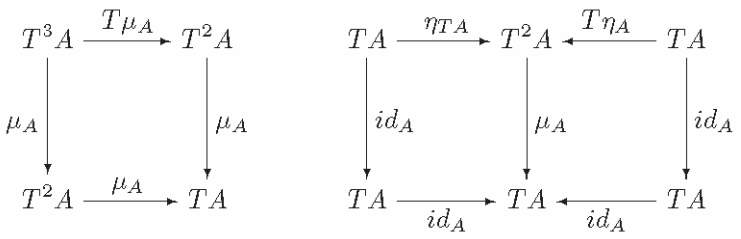
2 the objects of CT_{coalg} are pairs (A,h) with $A \in Ob_{DB}$ and $h \in DB(A,TA)$, the arrows between objects (A,h) and (B,k) are all arrows $f \in DB(A,B)$ such that $Tf \circ h = k \circ f : A \to TB$.

Definition 7: The monadic algebras/coalgebras, derived from a monad (T,η,μ), are defined (Asperti and Longo, 1991; Lane, 1971) as follows:

- Each T-algebra $(A,h:TA \to A)$, where h is a 'structure map', such that $h \circ \mu_A = h \circ Th$ and $h \circ \eta_A = id_A$ holds is a monadic T-algebra. The category of all monadic algebras T_{alg} is a full subcategory of CT_{alg}.
- Each T-coalgebra $(A,k:A \to TA)$, such that $Tk \circ k = \mu_A^C \circ k$ and $\eta_A^C \circ k = id_A$ holds is a monadic T-coalgebra. The category of all monadic coalgebras T_{coalg} is a full subcategory of CT_{coalg}.

Note: The monad (T,η,μ) given by commutative diagrams

Figure 2 Monads



defines the adjunction $< F^T, G^T, \eta^T, \mu^T >: DB \to T_{alg}$ such that $G^T \circ F^T = T : DB \to DB$, $\eta^T = \eta$, $\epsilon^T = \eta^{inv}$ and $\mu = G^T \epsilon^T F^T$. The functors $F^T : DB \to T_{alg}$ and $G^T : T_{alg} \to DB$ are defined as follows: for any object (database) A, $F^T(A) = (A, \eta_A^{inv} : TA \simeq A)$, while $G^T(A, \eta_A^{inv} : TA \simeq A) = TA$, for arrows F^T and G^T are identity functions.

Definition 8: Given a monad (T,η,μ) over a category $\mathcal{M}$, we have (Lane, 1971):

- *Kleisli triple* is a triple $(T,\eta,-^*)$, where for $f : A \to TB$ we have $f^* : TA \to TB$, such that the following equations hold: $\eta_A^* = id_{TA}$, $f^* \circ \eta_A = f$, $g^* \circ f^* = (g^* \circ f)$, for $f : A \to TB$ and $g : B \to TC$.

  A Kleisli triple satisfies the *mono requirement* provided η_A is monic for each object A.
- *Kleisli category* $\mathcal{M}_T$ has the same objects as $\mathcal{M}$ category. For any two objects A,B there is the bijection between arrows $\theta : \mathcal{M}(A,TB) \to \mathcal{M}(A,B)$. For any two arrows $f : A \to B$, $g : B \to C$ in $\mathcal{M}_T$, their composition is defined by $g \circ f \triangleq \theta(\mu_C \circ T\theta^{-1}(g) \circ \theta^{-1}(f))$.

The mono requirement for monad (T,η,μ) (Moggi, 1991) is satisfied because $\eta_A : A \to TA$ is a isomorphism $\eta_A = is_A$ (we denote its inverse by η_A^{-1}), thus it is also monic. Consequently, the category DB is a *computational model* for view-mappings (which are programs) based on observations (i.e., views) with the typed operator T, so that:

- TA is a *type of computations* (i.e., observations of the object of values A (of type A), which are the views of the database A).
- η_A is the *inclusion* of values into computations (i.e., inclusion of elements of the database A into the set of views of the database A). It is the isomorphism $\eta_A = is_A : A \to TA$.
- f^* is the 'equivalent' *extension* of a database mapping $f : A \to TB$, 'from values to computations' (programs correspond to call-by-value parameter passing) to a mapping 'from computations to computations' (programs correspond to call-byname), such that holds $f^* : Tf \to \mu_B \circ f \circ \eta_A^{-1}$, so $f^* \approx f$.

  Thus, in DB category, call-by-value $(f : A \to TB)$ and call-by-name $(f^* : TA \to TB)$ paradigms of programs are represented by *equivalent* morphisms $f \approx f^*$. Notice that in skeletal category DB_{sk} (which is equivalent to DB), all morphisms correspond to the call-by-name paradigm. This is because each arrow is a mapping from computations into computations (which are closed objects).

The basic idea behind the semantic of programs (Moggi, 1989) is that a program denotes a morphism from A (the object of *values* of type A) to TB (the object of *computations* of type B), according to the view of 'programs as functions from values to computations', so that the natural category for interpreting programs (in our case, a particular equivalent 'computation' database mappings of the form $f_1 \triangleq \eta_B \circ f : A \to TB$, derived from a database mapping $f : A \to B$, such that $f_1 \approx f$) is not a DB category but it is a Kleisli category DB_T.

In our case, the Kleisli category is a perfect model only for a subset of database mappings in DB: exactly for every view-mapping (i.e., query) $q_A : A \to TA$ which is just an arrow in Kleisli category $\theta(q_A) : A \to A$. For a general database mapping $f : A \to B$ in DB, only its (equivalent to f) 'computation extension' $\eta_B \circ f : A \to TB$ is an arrow $\theta(\eta_B \circ f) : A \to B$ in the Kleisli category. Consequently, the Kleisli category is a model for database mappings up to the equivalence '$\approx$'.

It means that, generally, database mappings are not simply programs from values into computations. In fact, the semantics of a database mapping between any two objects A and B can be specified as follows: for some set of computations (i.e., query-mappings) over A, we have the same equivalent (in the sense that these programs produce the same computed value (view)) set of computations (query-mappings) over B: it is fundamentally an *equivalence* of computations. This is a consequence of the fact that each database mapping (which *is not a function*) from A into B is naturally bidirectional, i.e., it is a morphism $f : A \to B$ and its equivalent reversed morphism $f^{inv} : B \to A$ *together* [explained by the duality property $DB = DB^{OP}$ (Majkić, 2008)]. Let us define this equivalence formally:

Definition 9: Each database mapping $h : A \to B$ is an equivalence of programs (epimorphisms), $h_A \triangleq \tau(J(h)) : A \twoheadrightarrow TH$ and $h_B \triangleq \tau^{-1}(J(h))^{inv} : B \twoheadrightarrow TH$ (τ and τ^{-1} are natural transformations of a categorial symmetry), where H generates a closed object $\tilde{h}$ (i.e., $TH = \tilde{h}$) and $h_A \approx h \approx h_B$, such that computations of these two programs (arrows of Kleisli category DB_T) are equal, i.e., $\partial_1(h_A) = \partial_1(h_B)$.

We can also provide an alternative model for equivalent computational extensions of database mappings in DB category:

Proposition 5: Denotational semantics of each mapping f, between any two database instances A and B, is given by the unique equivalent 'computation' arrow $f^1 \triangleq \eta_B \circ f$ in T_{coalg} from the monadic T-coalgebra (A,η_A) into a cofree monadic T-coalgebra (TB,μ_B^C), $f^1:(A,\eta_A)\to(TB,\mu_B^C)$, or , dually, by the unique equivalent arrow $f^1_{inv} \triangleq (\eta_B \circ f)^{inv} = f^{inv} \circ \eta_B^{inv}$ from the free monadic T-algebra (TB,μ_B) into the monadic T-algebra (A,η_A^{inv}).

Proof: In fact, $\mu_B^C \circ f = Tf \circ \eta_A$ holds. It is because $\mu_B^C = id_{TB}, \widetilde{\mu_B^C} \bigcap \tilde{f} = TB \bigcap \tilde{f} = \tilde{f}$ and $\widetilde{Tf} \bigcap \widetilde{\eta_A} = \widetilde{Tf} \bigcap TA = \widetilde{Tf} \bigcap TTA = \widetilde{Tf} \bigcap \tilde{f}$ (because $\tilde{f}$ is a closed object). □

Note that each view-map (query) $q_A : A \to TA$ is just equal to its denotational semantics arrow in T_{coalg}, $q_A:(A,\eta_A)\to(TA,\mu_B^C)$.

It is well-known that for a Kleisli category there exists an adjunction $< F^T, G^T, \eta^T, \mu^T >$ such that we obtain the same monad (T,η,μ), such that $T = G_T F_T, \mu = G_T \varepsilon_T F_T, \eta = \eta_T$. Let us see now how the Kleisli category DB_T is 'internalised' into the DB category.

Proposition 6: The Kleisli category DB_T of the monad (T,η,μ) is isomorphic to DB category, i.e., it may be 'internalised' in DB by the faithful forgetful functor $K = (K^0, K^1) : DB_T \to DB$, such that K^0 is an identity function and $K^1 \triangleq \phi\theta^{-1}$, where, for any two objects A and B:

- $\theta : DB(A,TB) \simeq DB_T(A,B)$ is Kleisli
- $\phi : DB(A,TB) \simeq DB(A,B)$, such that $\phi(\_) = \eta^{inv}_{cod(\_)} \circ \_$ is DB category bijection

respectively.

We can generalise a 'representation' for the base DB category (instead of usual Set category): a 'representation' of functor K is a pair $<\Upsilon,\varphi>, \Upsilon$ is the total object and $\varphi : DB_T(\Upsilon,\_) \simeq K$ is a natural isomorphism, where the functor $\varphi : DB_T(\Upsilon,\_) : DB_T \to DB$ defines 'internalised' hom-sets in DB_T, i.e., $DB^0_T(\Upsilon,B) \triangleq TB^{\Upsilon}$, $DB^1_T(\Upsilon,f) \triangleq id_\Upsilon \otimes Tf$.

Proof: Let us prove that ϕ is really a bijection in DB. For any program morphism $f : A \to TB$ we obtain $\phi(f) = \eta_B^{inv} \circ f : A \to B$ and, vice versa, for any $g : A \to B$ its inverse $\phi^{-1}(g) \triangleq \eta_B \circ g$, thus , $\phi\phi^{-1}(g) = (\eta_B \circ g) = \eta_B^{inv} \circ (\eta_B \circ g) = (\eta_B^{inv} \circ \eta_B) \circ g = id_B \circ g = g$ (because η_B is an isomorphism), i.e., $\phi\phi^{-1}$ is an identity function. Also $\phi^{-1}\phi(f) = \phi^{-1}(\eta_B^{inv} \circ f) = \eta_B \circ (\eta_B^{inv} \circ f) = (\eta_B \circ \eta_B^{inv}) \circ f = id_{TB} \circ f = f$, i.e., $\phi^{-1}\phi$ is an identity function, thus ϕ is a bijection.

Let us demonstrate that K is a functor: For any identity arrow $id_T = \theta(\eta_A) : A \to A$ in DB_T we obtain $K^1(id_T) = \phi\theta^{-1}(\theta(\eta_A)) = \phi(\eta_A) = \eta_A^{inv} \circ \eta_A = id_A$ (because η_A is an isomorphism). For any two arrows $g_T : B \to C$ and $f_T : A \to B$ in Kleisli category, we obtain, $K^1(g_T \circ f_T) = K^1(\theta(\mu_C \circ T\theta^{-1}(g_T) \circ \theta^{-1}(f_T))$ (from def. Kleisli category) $= \phi\theta^{-1}(\theta(\mu_C \circ Tg \circ f))$ (where $g \triangleq \theta^{-1}(g_T) : B \to TC$, $f \triangleq \theta^{-1}(f_T) : A \to TB) = \phi(g \circ \eta_B^{inv} \circ f)$ (it is easy to verify in DB that $\mu_C \circ Tg \circ f = g \circ \eta_B^{inv} \circ f) = \eta_C^{inv} \circ g \circ \eta_B^{inv} \circ f = \phi\theta^{-1}(\theta(g)) \circ \phi\theta^{-1}(\theta(f)) K^1(\theta\theta^{-1}(f_T)) = K^1(g_T) \circ K^1(f_T)$.

Thus, each arrow $f_T : A \to B$ in DB_T is 'internalised' in DB by its representation $f \triangleq K^1(f_T) = \phi\theta^{-1}(f_T) = \eta_B^{inv} \circ \theta^{-1}(f_T) : A \to B$, where $\theta^{-1}(f_T) : A \to TB$ is a program equivalent to the database mapping $f : A \to B$, i.e., $\theta^{-1}(f_T) \approx f$.

K is a faithful functor, in fact, for any two arrows $f_T, h_T : A \to B$ in DB_T, $K^1(f_T) = K^1(h_T)$ implies $f_T = h_T$: from $K^1(f_T) = K^1(h_T)$ we obtain $\phi\theta^{-1}(f_T) = \phi\theta^{-1}(h_T)$, if we apply a bijection $\theta\phi^{-1}$ we obtain $\theta\phi^{-1}(f_T) = \theta\phi^{-1}\phi\theta^{-1}(h_T)$, i.e., $\theta\theta^{-1}(f_T) = \theta\theta^{-1}(h_T)$, i.e., $f_T = h_T$ ($\theta\theta^{-1}$ and $\phi^{-1}\phi$ are identity functions). Let us prove that K is an isomorphism: from the adjunction $< F^T, G^T, \eta^T, \mu^T >: DB \to DB_T$, where F_T^0 is identity, $F_T^{-1} \triangleq \theta\phi^{-1}$, we obtain that $F_T \circ K = I_{DB_T}$ and $K \circ F_T = I_{DB}$, thus, the functor K is an isomorphism of DB and Kleisli category DB_T. □

Remark: It is easy to verify that a natural isomorphism $\eta : I_{DB} \to T$ of the monad (T, η, μ) is equal to the natural transformation $\eta : K \to G_T$. (consider that $G_T : DB_T \to DB$ is defined by, $G_T^0 = T^0$ and for any $f_T : A \to B$ in DB_T, $G_T^1(f_T) \triangleq \mu_B \circ T\theta^{-1}(f_T) : TA \to TB$).

Thus, the functor F_T has two different adjunctions: the universal adjunction $< F^T, G^T, \eta^T, \mu^T >$ which gives the same monad (T, η, μ) and this particular (for DB category only) isomorphism's adjunction $< F^T, K, \eta_I, \mu_I >$ which gives banal identity monad. We are now ready to define the semantics of queries in DB category and the categorical definition of *query equivalence*. This is important in the context of the Database integration/exchange and for the theory of *query-rewriting* (Halevy, 2000).

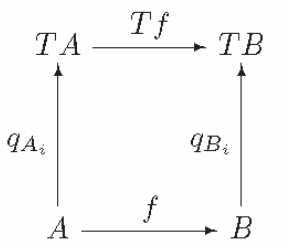
When we define a mapping (arrow, morphism) $f : A \to B$ between two databases A and B, implicitly we are defining the 'information flux' $\tilde{f}$, i.e., the set of views of A 'transmitted' by this mapping into B. Thus, in the context of query-rewriting, we consider only queries (i.e., view-maps) whose resulting view (observation) belongs to the 'information flux' of this mapping. Consequently, given any two queries, $q_{A_i} : A \to TA$ and $q_{B_j} : B \to TB$, they have to satisfy (w.r.t. query rewriting constraints) the condition $\partial_1(q_{A_i}) \in \tilde{f}$ (the $\partial_1(q_{A_i})$ is just a resulting view of this query) and $\partial_1(q_{B_j}) \in \tilde{f}$. So, the well-rewritten query over B, $q_{B_j} : B \to TB$, such that it is equivalent to the original query, i.e., $q_{B_j} \approx q_{A_i}$, must satisfy the condition $\partial_1(q_{B_j}) = \partial_1(q_{A_i}) \in \tilde{f}$.

Now we can give the denotational semantics for a query-rewriting in a data integration/exchange environment:

Proposition 7: Each database query is a (non-monadic) T-coalgebra. Any morphism between two T-coalgebras $f : (A, q_{A_i}) \to (B, q_{B_j})$ defines the semantics for relevant query-rewriting, when $\partial_1(q_{A_i}) \in \tilde{f}$.

Proof: Consider the following commutative diagram, where vertical arrows are T-coalgebras :

Figure 3 Queries



The morphism between two T-coalgebras $f:(A,q_{A_i})\to(B,q_{B_j})$ means that the commutativity $q_{B_j}\circ f = Tf\circ q_{A_i}: A\to TB$ is valid and from duality property we obtain $q_{B_i} = Tf\circ q_{A_i}\circ f^{inv}$. Consequently, for a given mapping $f: A\to B$ between databases A and B and for every query q_{A_i} such that $\partial_1(q_{A_i})\in\tilde{f}$ (i.e., $\widetilde{q_{A_i}}\subseteq\tilde{f}$), we can have an equivalent rewritten query q_{B_j} over a data base B. In fact $\widetilde{q_{B_j}} = \widetilde{Tf}\bigcap\widetilde{q_{A_i}}\bigcap\widetilde{f^{inv}} = \widetilde{q_{A_i}}$ because of the fact $\widetilde{q_{A_i}}\subseteq\tilde{f}$ and $\widetilde{f^{inv}} = \widetilde{Tf} = \tilde{f}$. Thus $q_{B_j}\approx q_{A_i}$. □

5 Conclusions

In this paper, we presented some fundamental properties and semantics for database mappings in the DB category. Majkic (2009) introduced the categorial (functors) semantics for two basic database operations: *matching* and *merging* (and data federation) and defined the algebraic database lattice. He has also shown that DB is concrete, small and locally finitely presentable (lfp) category and DB is also monoidal symmetric V-category enriched over itself. Based on these results, he developed a metric space and a subobject classifier for DB category and shown that it is a weak monoidal topos. In this paper, we considered some Universal algebra considerations and defined a categorical coalgebraic semantics for GLAV database mappings based on monads.

It was shown that a categorial semantics of database mappings can be given by the Kleisli category of the power-view monad T, that is, it was shown that Kleisli category is a model for database mappings up to the equivalence $\approx$ of morphisms in DB category. It was demonstrated that Kleisli category is isomorphic to the DB category and that call-by-values and call-by-name paradigms of programs (database mappings) are represented by equivalent morphisms. Moreover, it was shown that each database query (which is a program) is a monadic T-coalgebra and that any morphism between two T-coalgebras defines the semantics for the relevant query-rewriting.

References

Asperti, A. and Longo, G. (1991) *Categories, Types and Structures*, MIT Press.

Buneman, P., Naqui, S., Tanen, V. and Wong, L. (1995) 'Principles of programming with complex objects and collection types', *Theoretical Computer Science*, Vol. 149, No. 1.

Cali, A., Calvanese, D., Giacomo, G. and Lenzerini, M. (2003) 'Reasoning in data integration systems: why LAV and GAV are siblings', *Proceedings of the 14th International Symposium on Methodologies for Intelligent Systems, ISMIS 2003*, Springer 2871, pp.282–289.

Cohn, P.M. (1965) *Universal Algebra*, Harper and Row, London.

Halevy, A.Y. (2000) 'Theory of answering queries using views', *SIGMOD Record*, Vol. 29, No. 4, pp.40–47.

Kelly, G.M. and Power, A.J. (1993) 'Adjuntions whose counits are coequalizers, and presentations of finitary enriched monads', *J. Pure Appl. Algebra*, Vol. 89, pp.163–179.

Lambek, J. and Scott, P. (1986) *Introduction to Higher Order Categorical Logic*, Cambidge University Press.

Lane, S.M. (1971) *Categories for the Working Mathematician*, Springer-Verlag.

Lenzerini, M. (2002) 'Data integration: a theoretical perspective', in *Proc. of the 21st ACM SIGACT SIGMOD SIGART Symp. on Principles of Database Systems (PODS 2002)*, pp.233–246.

Majkić, Z. (1998) 'Categories: symmetry, n-dimensional levels and applications', PhD Thesis, University 'La Sapienza', Roma, Italy.

Majkić, Z. (2003a) 'The category-theoretic semantics for database mappings', Technical Report 14-03, University 'La Sapienza', Roma, Italy.

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

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

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

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

Moggi, E. (1989) 'Computational lambda-calculus and monads', in *Proc. of the 4th IEEE Symp. on Logic in Computer Science (LICS'89)*, pp.14–23.

Moggi, E. (1991) 'Notions of computation and monads', *Inf. and Comp.*, Vol. 93, No. 1, pp.55–92.

Plotkin, G.D. and Power, A.J. (2001) 'Adequacy for algebraic effects', *Proc. FOSSACS 2001, LNCS*, Vol. 2030, pp.1–24.

Walder, P. (1990) 'Comprehending monads', *Proceedings of ACM Conference on Lisp and Functional Programming*, Nice.