Synthesis of the Canonical Models for Database Integration Preserving Semantics of the Value Inventive Data Models

Leonid Kalinichenko, Sergey Stupnikov

Institute of Informatics Problems, Russian Academy of Science, Moscow, Russia
leonidandk@gmail.com, ssa@synth.ipi.ac.ru

Abstract. Recently families of sets of dependencies treated as the Datalog extensions were discovered for which the interpretation of queries becomes tractable. Such families are intended for inference of new, unknown values in the process of query answering. This paper considers such decidable classes of dependencies as the assets for creation of new data models (called in the paper the *value inventive* data models) analogously to axiomatic extension of the canonical model kernel used so far for unification of structured and object data models aimed at heterogeneous database integration. The paper[1] examines the decidable classes of dependencies reflecting the semantics of value inventive data models considering such classes to be the extensions of the canonical data model kernel. The issue of identifying of decidable subsets of the canonical model extension is considered.

Keywords: database integration, canonical data model synthesis, axiomatic data model extension, tuple-generating dependencies, decidable classes of dependencies, value inventive data models

1 Introduction

Throughout years in IT the trend of new information models (new languages) creation is stable. This trend is manifested in frame of emerging infrastructures (such as the OMG, Semantic Web, service-oriented, grid and cloud architectures) as well as during development of standards of specific languages, such as data models, semantic models, metadata and knowledge models in specific application areas, etc. This process is accompanied by another trend – by accumulation of information resources using such models the number of which is exponentially growing up. Such growth invokes the increasing demand for integrated usage of heterogeneous information resources defined in different information models as well as for their reuse and composition for implementation of interoperable information systems [18]. The demand for constructing of systems for virtual or

[1] This research has been done under the support of the RFBR (projects 10-07-00342-a, 11-07-00402-a) and the Program for basic research of the Presidium of RAS No. 16P (project 4.2).

materialized integration of information resources (further named I-systems for short) for problem solving in various application areas is huge and continues to grow fast.

Since generally in the I-systems the resources are heterogeneous (represented in different information models), for the homogeneous representation of their semantics it is required to reduce the diverse models to the uniform representation in frame of a unifying information model that is called the *canonical* one.

The main principle of the canonical model synthesis for the I-system consists in the extension of its kernel in the heterogeneous environment of a specific I-system. The kernel of the canonical model is fixed. For each specific information model M of the environment an extension of the kernel is defined so that the extension together with the kernel ensures preserving of semantics of information and operations representable in M. To reach that an extension of the kernel is specified in a declarative way to alter the kernel semantics in compliance with the M semantics. The canonical model is synthesized as a union of extensions defined for the models of the environment.

Investigations in the area of canonical models synthesis are continuing during a number of years. In the beginning the structured data models have been studying. The method for commutative data model mapping and axiomatic extension of the canonical model kernel as well as the templates of the family of axiom classes obtained for various structured data models appeared as a result of this period [19][20], denoted here as the *S-period*. Next period (*O-period*) of development of the methods for canonical model synthesis corresponded to the period of formation of the object and object-relational data models. Preserving of behavior in process of data model mapping required usage of formal metamodels providing for the proof of the refinement [1] of the data model specifications and their data types. Actually application of such metamodels can be considered as an elaboration [22] of the commutative structured data model mapping method with a retention of a possibility of usage of the axiomatic extension of the canonical model kernel. As a matter of fact, the axioms during the S-period as well as during the O-period acted the role of consistency constraints thus defining the semantics of data manipulation operations. Alongside with the studying of the canonical data models, during the O-period the possibilities of creation of canonical process models also have been studying for unification of various workflow models [24]. In this paper the consideration of the canonical model is limited with the *data models*.

Recently in the community of researchers in database and knowledge representation (with the influence of ontological models and the respective description logics) the studying of a new class of query languages intensified. Query answering for such languages uses reasoning based on the logical dependencies of data that results in inference of new values not existing in a database. The dependencies influencing the semantics of query languages are considered as new kinds of rules extending the Datalog language. Development of the Datalog$^{\pm}$ family [7][8][9] became an important result of this work. Query interpretation in such languages is based on the reasoning of their logical entailment of the base of

facts and such dependencies. Generally, this problem is undecidable. However, up to now, the sets of dependencies were defined [7][8] for which the problem of fixing of the query entailment becomes decidable and the interpretation of queries becomes tractable.

In the context of discussion of the canonical models and their synthesis it looks appropriate to consider such classes of dependencies as the assets for creation of new data models. In contrast to the axioms of the S- and O-periods of the development of methods of the canonical model synthesis that defined the semantics of the source model database update operations in the target model, the logical dependencies of new data models influence the semantics of the canonical model query language bringing features of inference into it. In particular, under such view, it is possible to use arbitrary query languages (e.g., SQL) over the database and decidable sets of dependencies. Thus, aiming at the database integration, query semantics can be expressed by a set of dependencies in the target schema adequately expressing query semantics of the source. Such view provides for extension of the diversity of data models considered for the practical data integration.

This paper starts a discussion of the third period (*I-period*) of development of the methods of the canonical model synthesis motivated by the appearance of new data models that will be called here as the *value inventive* data models. The inference of new, unknown values in the process of query answering is the distinguishing feature of these new data models. In course of this discussion it is assumed that the relational model is used as a kernel of the canonical data model and databases are considered as collections of relations on two distinct and disjoint domains: a set of constants and a set of *labeled nulls*. Labeled nulls ("fresh" Skolem terms, which are placeholders for unknown values, and can thus be seen as variables) are used during the generation of solutions to "invent" new values in the target database that do not appear in the source. One way to generate labeled nulls is through the Skolem functions [17].

Principles of database integration. Every data model is defined by the syntax and semantics of two languages — the data description language providing for definition of sets of admissible database states, and the data manipulation language providing for the definition of transformations of such states. The following propositions form the basis for a heterogeneous database integration concept [19][20].

The data model axiomatic extension principle. Canonical data model in the database integration system should be axiomatically extensible. Axiomatic in this context means that an extension of a target data model is carried out by adding to its DDL a set of axioms determining logical dependencies of the source data model in terms of the target model. Construction of a target DM axiomatic extension is considered as a new language design (DDL and DML) on the basis of the initial target data model languages.

Information and operation preserving principle. In the process of a source DM mapping into a canonical one it is necessary to preserve information and operations. Consider a mapping $f = \langle \sigma, \beta \rangle$ of a source data model M into extension

E of a canonical data model kernel C. Here σ maps schemas of M into schemas of E, β maps operations of DML of E into operation compositions of DML of M. The mapping f preserves information and operations if a bijective mapping θ between state spaces of schemas of M and E exists such that for every schema S of M and every operation o of DML of E the following condition holds. For every database states s_1 and s_2 admissible for S such that operation o transfers a database from the state $\theta(s_1)$ to the state $\theta(s_2)$, operation composition $\beta(o)$ transfers a database from the state s_1 to the state s_2.

The unifying canonical data model synthesis principle. Canonical data model synthesis is a process of construction of canonical data model kernel extensions preserving information and operations of the source data models of DBMSs embraced by the integrating system and merging of such extensions in a canonical data model. In such way a unifying canonical data model is formed in which data models of various DBMSs have unified representations (by the subsets of the canonical DM axiom set).

Value inventive data models. The recent extension of Datalog to existential rules [7] gave rise to discovery of various decidable classes of rules considered promising in an open-world perspective, where it cannot be assumed that all individuals are known in advance. In classical Datalog rules, variables are range-restricted, i.e., all variables in the rule head necessarily occur in a non-negated clause of the rule body. New kind of rules (containing in the body and in the head conjunctions of atoms) have an ability of value invention if they are existentially quantified in the head: $\forall x(Employee(x) \rightarrow \exists z(Manager(z, x) \wedge Employee(z))$. Application of such rule to a fact $Employee(E)$ where E is a constant invents new factual information $Manager(z_0, E) \wedge Employee(z_0)$. Here z_0 is a labeled null, a Skolem term, which is a placeholder for unknown values, and can thus be seen as a variable. Now our existential rule can be applied again to $Employee(z_0)$ producing another new Skolem term, and so on.

Such rules are known in databases as Tuple-Generating Dependencies (TGDs) [4]. Given a relational schema R, a *tuple-generating dependency* σ is a first-order formula of the form $\forall X, Y(\phi(X, Y) \rightarrow \exists Z(\psi(X, Z)))$, where $\phi(X, Y)$ and $\psi(X, Z)$ are conjunctions of atoms over R, called the body and the head of σ, respectively.

Conjunctive query answering under existentially quantified TGDs is undecidable, dependencies with differently restricted rule syntax for achieving decidability are the basic part of various value inventive classes of dependencies.

In the paper we provide an analysis of decidable classes of dependency templates considering them to be the axiomatic extensions of the canonical data model kernel for which the relational data model is assumed. Each such class reflects the semantics of a specific value inventive data model. On purely pragmatic reason we need to choose a way of identification of decidable collections of dependency templates making resulting classes of rules recognizable to check whether a given set of dependencies in a schema belongs to a certain class of dependencies. In [3] it was shown that abstract characterization of these classes based on schemes of the adequate reasoning mechanisms makes them not recognizable.

Therefore, definition of the classes by syntactic properties of the dependencies involved will be used here.

Each class is related to its role in the data integration setting or to a specific source data model which motivates definition of the class.

The paper starts with considering weakly-acyclic class of TGDs due to its importance in the data integration settings for GLAV schema mapping specification as a composition of two schema mappings. Then decidable classes of sets of dependencies belonging to the Datalog$^{\pm}$ family [8] are considered here as the extensions of the canonical model kernel preserving semantics of the respective ontological languages and description logics treated as data modeling formalisms [23]. In a separate section an extension of the canonical model kernel preserving semantics of the extended entity-relationship model is given. A short section is included to show how the extensions of the canonical model kernel constraining semantics of updates in accordance with the semantics of the respective structured data models are defined. The last section contains a discussion related to the issue of identifying of decidable subsets of the canonical model synthesized for a specific I-system.

2 Weakly-Acyclic Class of TGDs

Weakly acyclic sets of TGDs have been extensively studied in the context of data exchange (data integration) [14], where it was shown that they have tractable behavior solving the key algorithmic problems, including the existence of solutions, the computation of universal solutions[2] and conjunctive-query answering, and also the computation of the core of universal solutions. The chase[3] under weakly-acyclic sets of TGDs always terminates, and thus a finite instance C is constructed. Obviously query answering over C is decidable. Weakly-acyclic class (WAC) of TGDs is defined as follows [14].

WAC Syntactic Properties. Let Σ be a set of TGDs over a relational schema. Construct a directed *dependency graph* as follows:

1. Add a node for every pair (R, A) with R a relation symbol of the schema and A an attribute of R; call such pair (R, A) a *position*.
2. Add edges as follows. For every TGD $\varphi(X) \rightarrow \exists Y(\psi(X, Y))$ in Σ and for every x in X that occurs in ψ, for every occurrence of x in φ in position (R, A_i):

[2] The notion of query answering under dependencies is defined as follows. For a set of dependencies Σ on schema R, and a database D for R, the set of models (or solutions) of D given Σ, denoted $sol(D, \Sigma)$, is the set of all databases B such that $B \models D \cup \Sigma$. The set of answers to a CQ q on D given Σ, denoted as $ans(q, D, \Sigma)$, is the set of all tuples t such that $t \in q(B)$ for all $B \in sol(D, \Sigma)$. A solution $U \in sol(D, \Sigma)$ is *universal*, and we let $U \in usol(D, \Sigma)$, iff for all solutions $K \in sol(D, \Sigma)$, there is a homomorphism from U to K. A core solution might be considered as the "optimal" solution, in the sense that it is the universal solution of minimal size.

[3] Chase — is a process of repairing of a database D with respect to a set of dependencies Σ intended to get such resulted database that should satisfy these dependencies.

(a) for every occurrence of x in ψ in position (S, B_j), add an edge $(R, A_i) \rightarrow (S, B_j)$;
(b) in addition, for every existentially quantified variable y and for every occurrence of y in ψ in position (T, C_k), add a special edge $(R, A_i) \stackrel{*}{\rightarrow} (T, C_k)$.

Then Σ is weakly acyclic if the dependency graph has no cycles going through a special edge [14].

Sets of TGDs of weakly-acyclic class are used in a data integration (data exchange) setting, where we have a source schema S and a target schema T, assumed to be disjoint. Since T can be created independently of the source schema, it may have its own constraints that are given as a set Σ_t of sentences over T. The relationship between the source and target schemas is captured by source-to-target dependencies (Σ_{st}) that together with S and T specify source schema into target schema mapping. Actually, schema mapping is a schema (although partitioned in two parts) together with a set of constraints. Σ_{st} should be defined as a set of TGDs of weakly-acyclic class.

A data integration system is a triple (T, S, M), where T is the target (global) schema, S is the source schema, and M is a set of assertions relating elements of the target schema with elements of the source schema. It is assumed that source data model has been mapped into the canonical one, thus both source and target schemas are defined in the canonical data model. Both source and canonical data models may allow for the expression of various constraints. In this generality, a data integration setting is $\langle S, T, \Sigma_s, \Sigma_{st}, \Sigma_t \rangle$ in which S and Σ_s constitute the source schema, T and Σ_t form the target schema, and the source-to-target dependencies in Σ_{st} are the assertions of the schema mappings in the data integration system. Note that Σ_s includes not only data dependencies corresponding to the source database in the original data model, but also the dependencies generated in accordance with the extension of the kernel of the canonical data model determining an information-preserving mapping of the source data model into the canonical one.

The following GLAV setting characterization is considered here. Sound GLAV views are assumed [25]. GLAV mapping is constructed as a composition of two schema mappings — GAV mapping $M_{sg} = \langle S_s, S_g, \Sigma_{sg} \rangle$ (from source schemas to GAV global schema) and LAV mapping $M_{gt} = \langle S_g, S_t, \Sigma_{gt} \rangle$ (from GAV schema to target schema). A straightforward definition of the composition $M_{sg} \circ M_{gt}$ by means of the GAV and LAV dependencies follows. Thus, Σ_{st} consists of Σ_{sg} and Σ_{gt}, where the latter one is considered to be weakly-acyclic (it does not allow for cascading of labeled null creation during the chase).

- A GAV dependency is an s-g TGD of the form $\forall X(\varphi(X) \rightarrow \chi(X'))$ with $\varphi(X)$ a conjunction of atomic formulas over a source schema and $\chi(X')$ an atomic formula over a GAV schema such that the variables in X' are among those in X.
- A LAV dependency is an g-t TGD of the form $\forall X'(\chi(X') \rightarrow \exists Y(\psi(X'', Y)))$ with $\chi(X')$ an atomic formula over a GAV schema and $\psi(X'', Y)$ a conjunction of atoms over a target schema. X'' is a subset of X.

GLAV mappings are formed showing how two parts of the schema (source and target) are linked together by the Σ_{st} dependencies. Thus, the weakly-acyclic class of TGDs includes sets of full and embedded TGDs meeting the requirements of the WAC Syntactic Properties.

3 Decidable Classes of Dependencies of the Datalog$^{\pm}$ Family

Important family of decidable classes of sets of dependency templates has been defined under the Datalog$^{\pm}$ name. In particular, this research has been motivated by an intention to bring the results of research on data integration and exchange in databases to the context of the Semantic Web [7][8][9]. The purpose of inventing Datalog$^{\pm}$ family is to provide tractable query answering algorithms for more general languages than DL-Lite family of description logics. A brief overview of the classes of dependencies of the Datalog$^{\pm}$ family follows.

3.1 Guarded and Linear Classes

Guarded TGDs [8] is a class of TGDs (denoted as GC) relative to which query answering is decidable and tractable in the data complexity. A TGD σ is guarded iff it contains an atom (guard) in its body that contains all universally quantified variables of σ.

Example 1. The TGD $r(X,Y), s(Y,X,Z) \rightarrow \exists W s(Z,X,W)$ is guarded (via the guard $s(Y,X,Z)$), while the TGD $r(X,Y), r(Y,Z) \rightarrow r(X,Z)$ is not guarded.

Linear class of TGDs (LC) [8] is a subset of guarded class, where query answering is even FO-rewritable (can be reduced to evaluating a first order query over a relational database) in the data complexity. A TGD is linear iff it contains only a singleton body atom. Linear Datalog$^{\pm}$ generalizes the well-known class of inclusion dependencies, and is more expressive. For instance, the following linear TGD, which is not expressible with inclusion dependencies, asserts that everyone supervising her/himself is a manager: $supervises(X,X) \rightarrow manager(X)$.

Linear TGDs are generalized by multi-linear TGDs class (MLC). A TGD σ is multi-linear if each atom of body of σ is a guard. Each linear TGD is multi-linear since its single body-atom is automatically a guard. With multi-linear TGDs we can assert, for instance, that each employee who is also a manager supervises some other employee: $employee(X), manager(X) \rightarrow \exists Y supervises(X,Y)$. Clearly, this TGD is not linear.

In order to capture DL-Lite, the GC (LC) has been enriched by additional features: *negative dependencies and keys* [7]. A negative dependency is a Horn clause whose body is not necessarily guarded and whose head is the truth constant *false* denoted by $\bot$. For example, the requirement that a person ID cannot simultaneously appear in the $employee(ID, Name)$ and in the $retired(ID, Name)$ relation can be expressed by $employee(X,Y) \wedge retired(X,Z) \rightarrow \bot$.

An *equality-generating dependency* (EGD) is a formula η over R of the form $\forall X(\varphi(X) \rightarrow X_i = X_j)$ where $\varphi(X)$ is a conjunction of atoms and $X_i, X_j \in X$. It strictly generalizes key and functional dependencies.

A limited form of EGDs, namely, *key dependencies* (KD), are allowed to be specified, but it is required that these keys be not conflicting with the existential TGDs (the TGDs and the non-conflicting KDs do not interact, so that answers to queries can be computed by considering the TGDs only, and ignoring the KDs). A key dependency k is an assertion of the form $key(r) = A$, where r is a predicate symbol and A is a set of attributes of r. It is equivalent to the set of EGDs $\{r(X, Y_1, \ldots, Y_m), r(X, Y'_1, \ldots, Y'_m) \rightarrow Y_i = Y'_i\}_{1 \leq i \leq m}$, where the $X = X_1, \ldots, X_n$ appear exactly in the attributes in A.

Consider a set $\Sigma = \Sigma_T \cup \Sigma_K$ over R, where Σ_T and Σ_K are sets of TGDs and KDs, respectively. Let k be a key, and σ be a TGD of the form $\varphi(X, Y) \rightarrow \exists Z\ r(X, Z)$. Then, k is non-conflicting with σ [7] iff either (i) the relational predicate on which k is defined is different from r, or (ii) the positions of k in r are not a proper subset of the X-positions in r in the head of σ, and every variable in Z appears only once in the head of σ. We say k is non-conflicting (NC) with a set of TGDs Σ_T iff k is NC with every $\sigma \in \Sigma_T$. A set of keys Σ_K is non-conflicting (NC) with Σ_T iff every $k \in Sigma_K$ is NC with Σ_T.

Mapping of the DL-Lite axioms into the GC dependencies is given in Tab. 1.

Table 1. Rules of mapping of the DL-Lite axioms into the GC dependencies [16]

Dependency	DL-Lite Axiom	Dependencies in Datalog$^{\pm}$
Concept inclusion	$employee \sqsubseteq person$	$employee(X) \rightarrow person(X)$
(Inverse) Role inclusion	$reports- \sqsubseteq manager$	$reports(X, Y) \rightarrow manager(Y, X)$
Role transitivity	$trans(manager)$	$manager(X, Y), manager(Y, Z) \rightarrow manager(X, Z)$
Participation	$employee \sqsubseteq \exists report$	$employee(X) \rightarrow \exists Y report(X, Y)$
Disjointness	$employee \sqsubseteq \neg customer$	$employee(X), customer(X) \rightarrow \bot$
Functionality	$funct(reports)$	$reports(X, Y), reports(X, Z) \rightarrow Y = Z$

3.2 Sticky Class of TGDs

The class of *sticky* sets of TGDs (SC) [7][15] imposes a restriction on multiple occurrences of variables in the rule bodies. SC is defined by a sufficient syntactic condition that ensures the so-called *sticky property* of the chase. The definition of sticky sets of TGDs is based on a variable-marking procedure SMarking. This procedure accepts as input a set of TGDs Σ, and marks the variables that occur in the body of the TGDs of Σ. Formally, SMarking(Σ) works as follows. First, we apply the so-called *initial marking* step: for each TGD $\sigma \in \Sigma$, and for each variable v in $body(\sigma)$, if there exists an atom a in $head(\sigma)$ such that v does not appear in a, then we mark each occurrence of v in $body(\sigma)$. Then, we apply exhaustively (i.e., until a fixpoint is reached) the *propagation* step: for each pair

of TGDs $\langle \sigma, \sigma' \rangle \in \Sigma \times \Sigma$ (including the case $\sigma = \sigma'$), if a $\forall$-variable v occurs in $head(\sigma)$ at positions $\pi_1, \ldots, \pi_m$, for $m \geq 1$, and there exists an atom $a \in body(\sigma)$ such that at each position $\pi_1, \ldots, \pi_m$ a marked variable occurs, then we mark each occurrence of v in $body(\sigma)$.

SC Syntactic Properties. A set Σ of TGDs is *sticky* if there is no TGD $\sigma \in SMarking(\Sigma)$ such that a marked variable occurs in $body(\sigma)$ more than once.

Example 2. The following relational schema is used from [9]:

$dept(DeptId, MgrId)$	$in\_area(ProjectId, Area)$
$emp(EmpId, DeptId, Area, ProjectId)$	$project\_mgr(EmpId, ProjectId)$
$runs(DeptId, ProjectId)$	$external(ExtId, Area, ProjectId)$

The following set Σ of TGDs for this example is a sticky set:
$dept(V, W) \rightarrow \exists X, Y, Z\ emp(W, X, Y, Z)$.
$emp(V, W, X, Y) \rightarrow \exists Z\ dept(W, Z), runs(W, Y), in\_area(Y, X)$.
$runs(W, X), in\_area(X, Y) \rightarrow \exists Z\ external(Z, Y, X)$.

Sticky sets are FO-rewritable and can be used with relational database schemas of *arbitrary arity* (in contrast, the DL-Lite languages are usable for binary relations only) [11]. Sticky sets of TGDs can express constraints and rules involving *joins*. Since query answering under TGDs involving joins is undecidable in general, it is required to restrict the interaction of TGDs, when joins are used.

3.3 Weakly-Sticky Class of TGDs

Weakly-sticky class of TGD (WSC) [9] generalizes both weakly-acyclic and sticky classes of TGDs.

WSC Syntactic Properties. In a weakly-sticky set of TGDs, the variables that occur more than once in the body of a TGD are either non-marked, or they occur at positions where a finite number of distinct values can appear during the chase. Strict definition is given in [9].

Consider the set Σ of TGDs over the schema in Ex. 2:
$dept(V, W) \rightarrow \exists X, Y\ emp(W, V, X, Y)$.
$emp(V, W, X, Y) \rightarrow \exists Z\ dept(W, Z), runs(W, Y)$.
$runs(W, X), dept(W, Y) \rightarrow project\_mgr(Y, X)$.

3.4 Sticky-Join Class of TGDs

Despite their expressiveness, sticky sets of TGDs are not powerful enough to be able to capture simple cases such as the TGD $r(X, Y, X) \rightarrow \exists Z\ s(Y, Z)$. Clearly, the variable X is marked, and thus stickiness is violated. Notice that the above TGD is linear. A first-order rewritable class which captures both sticky sets of TGDs and linear TGD is called *sticky-join* class of TGDs (SJC) [9]. It allows for a limited form of join (including as special case the Cartesian product), providing for expressing of the natural ontological relationships not expressible in OWL.

SJC Syntactic Properties. The main disadvantage of this class is the fact that the identification problem, i.e., whether a set of TGDs is sticky-join, is computationally hard; in particular, PSPACE-hard. Notice that the identification problem under (multi-)linear TGDs and sticky sets of TGDs is feasible in PTIME.

Consider the following set Σ of TGDs that is sticky-join, but not sticky, not weakly sticky and not weakly-acyclic [9]:
$r(X_1, Y_1), p(Z_1, W_1) \rightarrow s(X_1, Y_1, Z_1, W_1)$.
$s(X_2, Y_2, Z_2, Z_2) \rightarrow \exists W_2\ r(W_2, Y_2), r(X_2, W_2)$.

Sticky-join sets of TGDs are FO-rewritable. Analogously to weakly-sticky sets of TGDs, it is possible to define the class of weakly-sticky-join class of TGDs (WSJC), that generalizes both weakly-acyclic and sticky-joins classes of TGDs.

3.5 Preserving of the DL-Lite Description Logics Semantics by the Relational Model Extended with the Classes of Dependencies of the Datalog$^{\pm}$ Family

The main DL-Lite languages, namely, DL-Lite$_F$, DL-Lite$_R$ and DL-Lite$_A$, can be mapped to linear TGDs and sticky sets of TGDs, combined with negative constraints (NCs) and EGDs without loosing FO-rewritability, and consequently highly tractability of query answering in data complexity. Moreover, the DLs DL-Lite$_{F,\sqcup}$, DL-Lite$_{R,\sqcup}$ and DL-Lite$_{A,\sqcup}$ obtained from DL-Lite$_F$, DL-Lite$_R$ and DL-Lite$_A$, respectively, by additionally allowing conjunction in the left-hand side of the axioms, can be reduced to multi-linear TGDs (with NCs and KDs). Furthermore, the above DLs (with binary roles) have a counterpart in the DLR-Lite family, which allows for n-ary roles, along with suitable constructs to deal with them [6]. These extended languages can be also reduced to (multi-)linear and sticky class of TGDs (with NCs and KDs) [16].

4 Class of Dependencies Extending Relational Model to the Semantics of the Augmented Entity-Relationship Model

An Entity-Relationship formalism augmented with such features as is-a among entities and relationships, mandatory and functional participation of entities to relationships, mandatory and functional attributes of entities and relationships is overviewed here as an example of yet another value inventive data model that is called $\mathcal{ER}^+$ [10][5][12]. Treating the relational data model as a canonical one, the overview will be focused on a set of dependencies Σ extending the relational model semantics up to that of $\mathcal{ER}^+$. This set is based on the KD and TGD types (actually, the TGDs required are *inclusion dependencies* considered as a special class of TGDs [2]). Mapping of $\mathcal{ER}^+$ schema into the extended relational model is considered. In [7] it is shown that to make conjunctive queries in the extended

relational model FO-rewritable under Σ an additional syntactic condition on the respective dependencies in Σ should be imposed to guarantee *separation*, i.e., the absence of interaction between KDs and TGDs. $\mathcal{ER}^+$ provides for querying of the incomplete source databases with value invention mechanism similar to description logics.

An $\mathcal{ER}^+$ schema consists of a collection of entity, relationship, and attribute definitions over a set of entity symbols (denoted by Ent), a set of relationship symbols (denoted by Rel), and a set of attribute symbols (denoted by Att).

Example 3. Consider the $\mathcal{ER}^+$ schema C defined as follows [5].

$entity\ Employee$
  $participates(\geq 1) : Works\_in : 1$
  $participates(\leq 1) : Works\_in : 1$
$entity\ Manager\ isa:\ Employee$
  $participates(\leq 1):\ Manages : 1$
  $participates(\geq 1):\ Manages : 1$
$entity\ Dept$
$relationship\ Works\_in\ among\ Employee,\ Dept$
$relationship\ Manages\ among\ Manager,\ Dept\ isa:\ Works\_in[1,2]$
$attribute\ emp\_name\ of\ Employee$
$attribute\ dept\_name\ of\ Dept$
$attribute\ since\ of\ Works\_in$

An *entity definition* includes (i) the *isa* clause specifies a set of entities to which E is related via is-a; (ii) the $participates(\geq 1)$ clause specifies those relationships in which an instance of E must necessarily participate; and for each relationship R_i, the clause specifies that E participates as c_i-th component in R_i; (iii) the $participates(\leq 1)$ clause specifies those relationships in which an instance of E cannot participate more than once.

A *relationship definition* includes (i) the n entities of Ent, with $n \geq 2$, listed in the *among* clause are those among which the relationship is defined (i.e., component i of R is an instance of entity E_i); (ii) the *isa* clause specifies a set of relationships to which R is related via is-a; for each relation R_i, in square brackets it is specified how the components $[1, \ldots, n]$ are related to those of e_i, by specifying a permutation $[j_{i1}, \ldots, j_{in}]$ of the components of E_i; (iii) the number n of entities in the among clause is the *arity* of R. The *isa* clause is optional.

An *attribute definition* includes (i) the entity or relationship X with which the attribute is associated; (ii) *qualification* consists of none, one, or both of the keywords *functional* and *mandatory*, specifying respectively that each instance of X has a unique value for attribute A, and that each instance of X needs to have at least a value for attribute A. If the functional or mandatory keywords are missing, the attribute is assumed by default to be multivalued and optional, respectively.

$\mathcal{ER}^+$ schema C is mapped into the relational schema S as a set of *relational symbols* or *predicates*, each with its associated arity. r/n denotes that the predi-

cate r has arity n. A *position* $r[i]$ (in a schema S) is identified by a predicate $r \in S$ and its i-th argument (or attribute). Entity, attribute, relationship, attribute of a relationship are mapped into unary, binary, n-ary, $n+1$-ary predicates, respectively. $e(c)$, $a(c,d)$, $r(c_1,\ldots,c_n)$, $a(c_1,\ldots,c_n,d)$ denote an instance of entity E (with c as a surrogate key), d as a value of attribute A for an entity E, an instance of relationship R, d as a value of attribute A associated with the instance of relationship R, respectively. The intended semantics of $\mathcal{ER}^+$ data model is captured by extending of the relational model with the class of dependencies preserving $\mathcal{ER}^+$ semantics (Tab. 2 [12]).

Table 2. Correspondence of relational dependencies and $\mathcal{ER}^+$ constructs

$\mathcal{ER}^+$ **Construct**	**Class of relational dependencies**
attribute A for an entity E	$a(X,Y) \rightarrow e(X)$
attribute A for a relationship R	$a(X_1,\ldots,X_n,Y) \rightarrow r(X_1,\ldots,X_n)$
relationship R with entity E as i-th component	$r(X_1,\ldots,X_n) \rightarrow e(X_i)$
mandatory attribute A of entity E	$e(X) \rightarrow \exists Y\ a(X,Y)$
mandatory attribute A of relationship R	$r(X_1,\ldots,X_n) \rightarrow \exists Y\ a(X_1,\ldots,X_n,Y)$
functional attribute A of an entity	$key(a) = \{1\}$(a has arity 2)
functional attribute A of a relationship	$key(a) = \{1,\ldots,n\}$(a has arity $n+1$)
is-a between entities E_1 and E_2	$e_1(X) \rightarrow e_2(X)$
is-a between relationships R_1 and R_2 where components $1,\ldots,n$ of R_1 correspond to components $i_1,\ldots,i_n$ of R_2	$r_1(X_1,\ldots,X_n) \rightarrow r_2(X_{i1},\ldots,X_{in})$
mandatory participation of E in R (i-th component)	$e(X) \rightarrow r(X_1,\ldots,X_{i-1},X,X_{i+1},\ldots,X_n)$
functional participation of E in R (i-th component)	$key(r) = \{i\}$

The set Σ of dependencies over relational schema S associated to the $\mathcal{ER}^+$ schema C of the Ex. 3 is the following:

$manager(X) \rightarrow employee(X)$
$since(X,Y,Z) \rightarrow works\_in(X,Y)$
$deptname(X,Y) \rightarrow dept(X)$
$emp\_name(X,Y) \rightarrow employee(X)$
$manages(X,Y) \rightarrow works\_in(X,Y)$
$employee(X) \rightarrow \exists Y works\_in(X,Y)$
$key(works\_in) = 1$
$manager(X) \rightarrow \exists Y\ manages(X,Y)$
$key(manages) = \{1\}$
$works\_in(X,Y) \rightarrow employee(X)$
$works\_in(X,Y) \rightarrow dept(Y)$
$manages(X,Y) \rightarrow manager(X)$
$manages(X,Y) \rightarrow dept(Y)$

Introducing the non-conflicting keys condition for $\mathcal{ER}^+$ requires the notion of Σ-*graph*. Consider a set Σ of dependencies over a schema S. The Σ-*graph* for S and Σ is defined as follows: (i) the set of nodes is the set of positions in S; (ii)

if there is a TGD σ in Σ such that the same variable appears in a position p_b in the body and in a position p_h in the head, then there is an arc from p_b to p_h.

A node corresponding to a position derived from an entity (resp., a relationship) is called an *e-node* (resp., an *r-node*). Moreover, an r-node corresponding to a position which is a unary key in a relationship is called a *k-node*.

Non conflicting Σ. Consider a set Σ of dependencies over a schema S, and let G be the Σ-graph for S and Σ. Σ is said to be *non-conflicting* if the following condition is satisfied: for each path $v_1 \curvearrowright v_2 \curvearrowright \cdots \curvearrowright v_m$ in G, where $m \geq 3$, such that: (i) v_1 is an e-node, (ii) $v_2, \ldots, v_{m-1}$ are r-nodes, and (iii) v_m is a k-node, there exists a path in G of only r-nodes from v_m to v_2.

$\mathcal{ER}^+$ Class Syntactic Properties. Rules in the sets of belonging to the $\mathcal{ER}^+$ class of relational dependencies should be non-conflicting and should conform to their templates given in the Tab. 2.

The classes of dependencies in the Datalog$^\pm$ family are more expressive (and less tractable) than $\mathcal{ER}^+$ class except for *Linear Class* that allows for query answering in AC_0 in data complexity. However, the class of non-conflicting $\mathcal{ER}^+$ dependencies is not expressible in LC with its definition of the of non-conflicting KDs [7]). It means that though dependencies look similarly, different syntactic constraints on them are imposed making the extensions different. Non-conflicting $\mathcal{ER}^+$ is strictly more expressive than the languages DL-Lite$_F$ and DL-Lite$_R$.

5 Axiomatic Extensions of Relational Data Model Preserving Semantics of the Structured Data Models

For completeness of examinations of various classes of axiom sets extending the relational model as a canonical one we provide a brief reference to the classes of constraints defined during the S-period of canonical model synthesis (Sect. 1). In [20] applying the data model mapping principles summarized in Sect. 1 the axiomatic extensions of relational data model preserving semantics of various source structured data models (e.g., network, hierarchical, binary, etc.) were presented. Axiomatic extensions of relational model in S-period were targeted at the adequate modification of the DML update operations semantics to preserve the axioms of the extension. To emphasize such behavioral orientation of the axioms introduced in the S-period, they are called here constraints. The family of 12 classes of constraint sets for different data models was introduced [20]. Each class contained a subset of the 18 axioms defined for the structured models of that period. In the Tab. 3 we show a subset of the class of constraints preserving semantics of the CODASYL network data model in the form of full TGDs and KDs.

A set of similar axiom templates was reused during the O-period of the canonical model synthesis: it was included as a part of the frame-based, object-oriented canonical model kernel (SYNTHESIS language [21]).

Table 3. Constraints preserving semantics of the CODASYL network data model

CODASYL Constraint	**TGD or KD form**
Constraint of uniqueness of values of attributes $A_1, \ldots, A_n$ of a relation R	$key(R): \{A_1, \ldots, A_n\}$.
Constraint of optional uniqueness of values of attributes $A_1, \ldots, A_n$ of relation R	$key\_nonnull(R): \{A_1, \ldots, A_n\}$.
Constraint of definiteness of values of attributes $A_1, \ldots, A_n$ of relation R	$R(A_1, \ldots, A_n),\ null(A_1, \ldots, A_n) \rightarrow \bot$.
CODASYL set with mandatory members participation (R_i, R_j are relational images of owner (member) record types, respectively)	$R_j(A_1, \ldots, A_n) \rightarrow R_i(A_1, \ldots, A_n)$. $key(R_i): A_1, \ldots, A_n$.
CODASYL set with optional members participation (R_i, R_j are relational images of owner (member) record types, respectively)	$R_j(A_1, \ldots, A_n),\ nonnull(A_1, \ldots, A_n.) \rightarrow R_i(A_1, \ldots, A_n)$. $key(R_i): \{A_1, \ldots, A_n\}$.

6 Decidable Subsets of the Canonical Model Kernel Extension

In the S- and O-periods the canonical model synthesis was reduced to the process of construction of canonical data model kernel extensions preserving semantics of various source data models and merging of such extensions in a canonical data model. Constraints introduced in that periods did not influence the canonical query language semantics. But in the I-period the decidable extensions (classes of dependencies) preserving semantics of query languages of various source data models are constructed. Such classes of dependencies presented in Sect. 3 and 4 can be merged in the canonical data model. It means that in the target schemas, taking dependencies from such merged canonical model, it might be required to include the combinations of such dependencies belonging to several extension classes. These classes are recognizable by their syntactic properties (Sect. 3, 4).

A query can refer to relations so that a combination of the dependencies related to them could belong to different decidable classes. It is required to reason whether the union of two or more decidable classes remains to be decidable. In [3][26] it is shown that the unjustifiable union of two sets of dependencies belonging to different decidable classes can lead to undecidability. The study of conditions on the interaction between dependencies allowing for their safe union has been attempted by means of the graph of rule dependencies GRD [26]. As a result, an *inclusion relationship* between various decidable classes of dependencies has been defined. Classes of dependencies possessing value inventive capabilities overviewed in this paper are related through the inclusion relationships (denoted by $\lesssim$) shown on Fig. 1.

Decidable classes belonging to one path are compatible and can be safely merged. Further investigations are required to clarify a possibility of merging classes belonging to different paths.

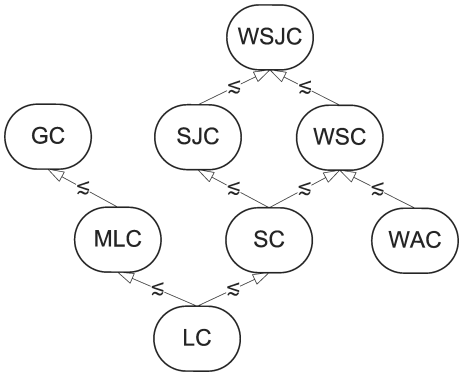

Fig. 1. Dependency classes hierarchy

7 Conclusion

This paper presents an approach to the canonical model synthesis methods for new kind of source data models that are called in the paper "the *value inventive* data models", thus stressing that inference of new, unknown values in the process of query answering is the distinguishing feature of these data models. Recently families of sets of dependencies were defined [7][8][9] for which the problem of fixing of the query entailment from such dependencies becomes decidable and the interpretation of queries becomes tractable.

In the context of discussion of the canonical models and their synthesis for the database integration the paper considers such classes of dependencies as the assets for creation of new data models analogously to axiomatic extension of the canonical model kernel used so far for integration of structured and object data models. In accordance with this approach, the kernel of the canonical model (here the relational data model is assumed) is extended with the classes of the dependency templates, each class providing for expression in the extended data model the semantics of a source data model. For unification of the value inventive data models the classes of dependencies extending the canonical model kernel reflect the alteration of the semantics of the query language of the canonical model kernel bringing features of inference into it.

In the paper we examine the decidable classes of dependencies reflecting the semantics of value inventive data models considering such classes to be extensions of the canonical data model kernel. The classes of dependencies that were analyzed in the paper include weakly-acyclic class of TGDs due to its importance in the data integration settings for specifying GLAV schema mappings, decidable classes of sets of dependencies belonging to the Datalog$^{\pm}$ family considered here as the extensions of the canonical model kernel preserving semantics of the respective ontological languages and description logics treated here as data modeling formalisms [23], as well as the extension of the canonical model kernel preserving semantics of the augmented entity-relationship data model. The pa-

per emphasizes syntactic properties of each class of dependencies as the means for recognizing of belonging of a specific set of dependencies to the respective class.

The following value inventive data models (including data models that can be based on the respective description logics) can be presented in the canonical model in a uniform way as its extensions with the classes of dependencies considered in the paper: description logics of the DL-Lite family (including DL-Lite$_F$, DL-Lite$_R$, DL-Lite$_A$, DL-Lite$_{F,\sqcup}$, DL-Lite$_{R,\sqcup}$ and DL-Lite$_{A,\sqcup}$), OWL 2 QL (based on DL-Lite$_R$), DLR-Lite, $\mathcal{ER}^+$ data model. Similarly, various structured data models can be represented in the canonical model by the respective classes of constraints as it is shown in Sect. 5.

The challenging problems related to the study of the canonical models unifying different value inventive data models include:

- Deep analysis of relationships between various classes of dependencies to justify possibilities of getting their decidable merge.
- Investigating methods for rewriting of queries over a target schema in the GLAV setting containing sets of dependencies belonging to different classes. New requirements to query rewriting include a necessity to recognize a class (classes) of dependencies related to the query and to reason whether such combination of rules is decidable. Each class might require a specific query rewriting algorithm.
- Study of approaches to constructing a target schema in the GLAV setting applying various dependencies of the canonical model to meet the requirements of the problem domain (note that Σ_t is defined independently of the sources and their data models); study of the approaches to GLAV mapping specification depending on the interrelationship between Σ_t and Σ_s.

References

1. J.-R. Abrial. The B-Book. Cambridge University Press, 1996.
2. S. Abiteboul, R. Hull, and V. Vianu. Foundations of Databases. Addison-Wesley, 1995.
3. J.-F. Baget, M. Leclere, M.-L. Mugnier. Walking the Decidability Line for Rules with Existential Variables. In Proceedings of the 12th International Conference on the Principles of Knowledge Representation and Reasoning (KR 2010).
4. Beeri C., Vardi M.Y. A proof procedure for data dependencies. JACM 31(4), 1984, 718-741.
5. Andrea Cali, Davide Martinenghi. Querying Incomplete Data over Extended ER Schemata. Theory and Practice of Logic Programming, Volume 10 - Special Issue 03 (Logic Programming in Databases: from Datalog to Semantic-Web Rules), Cambridge University Press 2010, 291-329.
6. Calvanese D., Giacomo G. D., Lembo D., Lenzerini M., Rosati R. Data complexity of query answering in description logic. In Proc. of KR, 2006, pp. 260-270.
7. A. Cali, G. Gottlob, T. Lukasiewicz. A General Datalog-Based Framework for Tractable Query Answering over Ontologies. PODS09, June 29July 2, 2009.

8. Andrea Cali, Georg Gottlob, Thomas Lukasiewicz, Bruno Marnette, Andreas Pieris. Datalog$^{\pm}$: A Family of Logical Knowledge Representation and Query Languages for New Applications. 25th Annual IEEE Symposium on Logic in Computer Science, 2010. pp. 228-242.
9. Andrea Cali, Georg Gottlob, and Andreas Pieris. Query Answering under Non-guarded Rules in Datalog$^{\pm}$. RR 2010, LNCS 6333, pp. 1-17, 2010.
10. Andrea Cali, Georg Gottlob and Andreas Pieris. Tractable Query Answering over Conceptual Schemata. ER 2009, LNCS 5829, pp. 175-190, 2009
11. Andrea Cali, Georg Gottlob and Andreas Pieris. New Expressive Languages for Ontological Query Answering. Proc. of the Twenty-Fifth AAAI Conference on Artificial Intelligence, 2011.
12. Andrea Cali, Georg Gottlob, AndreasPieris. Ontological query answering under expressive EntityRelationship schemata. Information Systems, 37, 2012, pp. 320-335
13. Balder ten Cate and Phokion G. Kolaitis. Structural Characterizations of Schema-Mapping Languages. CACM, 2010, vol. 53, no. 1.
14. R. Fagin, P. G. Kolaitis, R. J. Miller, L. Popa. Data exchange: semantics and query answering. Theoretical Computer Science 336 (2005), 89-124.
15. Georg Gottlob, Giorgio Orsi, and Andreas Pieris. Ontological Query Answering via Rewriting. ADBIS 2011, LNCS 6909, pp. 1-18, 2011.
16. G.Gottlob, Giorgio Orsi, Andreas Pieris. Ontological Queries Rewriting and Optimization, ICDE-2011.
17. Richard Hull, Masatoshi Yoshikawa. ILOG: Declarative Creation and Manipulation of Object Identifiers. VLDB 1990, p. 455-468
18. Kalinichenko L. A., Briukhov D. O., Martynov D.O., Skvortsov N. A., Stupnikov S. A. Mediation framework for enterprise information system infrastructures. 9th International Conference on Enterprise Information Systems (ICEIS), 2007.
19. Kalinichenko L. A. Methods and Tools for Integration of Heterogeneous Databases. Moscow, Science Publ., 1983, 423p. (in Russian).
20. Kalinichenko L. A. Methods and tools for equivalent data model mapping construction. Proc. of the EDBT'90 Conference, 1990, Springer Verlag, p.92-119.
21. Kalinichenko L. A. SYNTHESIS: the language for description, design and programming of the heterogeneous interoperable information resource environment. Institute of Informatics Problems, Russian Academy of Sciences, Moscow, 1993.
22. Kalinichenko L. A. Method for Data Models Integration in the Common Paradigm. Proceedings of the First East-European Conference ADBIS'97, St.Petersburg, 1997.
23. Kalinichenko L. A., Stupnikov S. A. OWL as Yet Another Data Model to be Integrated. Proc. ADBIS 2011. - Vienna: Austrian Computer Society, 2011. - P. 178-189.
24. Kalinichenko L.A., Stupnikov S.A., Zemtsov N.A. Extensible Canonical Process Model Synthesis Applying Formal Interpretation. Proc. ADBIS 2005. LNCS 3631. – Berlin-Heidelberg: Springer-Verlag, 2005. – P. 183-198.
25. M. Lenzerini. Data Integration: a Theoretical Perspective. In PODS, pages 233-246, 2002.
26. Marie-Laure Mugnier. Ontological Query Answering with Existential Rules. RR 2011, LNCS 6902, pp. 2-23, 2011