Query rewriting using views in a typed mediator environment

Leonid A. Kalinichenko, Dmitry O. Martynov, Sergey A. Stupnikov

Institute of Informatics Problems, Russian Academy of Sciences,
Vavilov Street, 44 building 2, 119333, Moscow, Russian Fed.
`{leonidk,domart,ssa}@synth.ipi.ac.ru`

Abstract. Query rewriting method is proposed for the heterogeneous information integration infrastructure formed by the subject mediator environment. Local as View (LAV) approach treating schemas exported by sources as materialized views over virtual classes of the mediator is considered as the basis for the subject mediation infrastructure. In spite of significant progress of query rewriting with views, it remains unclear how to rewrite queries in the typed, object-oriented mediator environment. This paper embeds conjunctive views and queries into an advanced canonical object model of the mediator. The “selection-projection-join” (SPJ) conjunctive query semantics based on type specification calculus is introduced. The paper demonstrates how the existing query rewriting approaches can be extended to be applicable in such typed environment. The paper shows that refinement of the mediator class instance types by the source class instance types is the basic relationship required for establishing query containment in the object environment.

1 Introduction

This work has been performed in frame of the project [12] aiming at building large heterogeneous digital repositories interconnected and accessible through the global information infrastructure[1]. In this infrastructure a middleware layer is formed by subject mediators providing a uniform ontological, structural, behavioral and query interface to the multiple data sources. In a specific domain the subject model is to be defined by the experts in the field independently of relevant information sources. This model may include specifications of data structures, terminologies (thesauri), concepts (ontologies), methods applicable to data, processes (workflows), characteristic for the domain. These definitions constitute specification of a subject mediator. After subject mediator had been specified, information providers can register their information at the mediator for integration in the subject domain. Users should know only subject domain definitions that contain concepts, data structures, methods approved by the subject domain community. Thus various information sources belonging to different providers can be registered at a mediator. The subject mediation is applica-

[1] This work has been partially supported by the grants of the Russian Foundations for Basic Research N 03-01-00821 and N 04-07-90083

ble to various subject domains in science, cultural heritage, mass media, e-commerce, etc.

Local as View (LAV) approach [8] treating schemas exported by sources as materialized views over virtual classes of the mediator is considered as the basis for the subject mediation infrastructure. This approach is intended to cope with dynamic, possibly incomplete set of sources. Sources may change their exported schemas, become unavailable from time to time. To disseminate the information sources, their providers register them (concurrently and at any time) at respective subject mediators. A method and tool supporting process of information sources registration at the mediator were presented in [1]. The method is applicable to wide class of source specification models representable *in hybrid semi-structured/object canonical mediator model*. Ontological specifications are used for identification of mediator classes semantically relevant to a source class. A subset of source information relevant to the *mediator classes* is discovered based on identification of maximal commonality between a source and mediated level class specification. Such commonality is established so that compositions of mediated class instance types could be *refined* by a source class instance type.

This paper (for the same infrastructure as in [1]) presents an approach for query rewriting in a typed mediator environment. The problem of rewriting queries using views has recently received significant attention. The data integration systems described in [2,13] follow an approach in which the contents of the sources are described as views over the mediated schema. Algorithms for answering queries using views that were developed specifically for the context of data integration include the Bucket algorithm [13], the inverse-rules algorithm [2,3,15], MiniCon algorithm [14], the resolution-based approach [7], the algorithm for rewriting unions of general conjunctive queries [17] and others.

Query rewriting algorithms evolved into conceptually simple and quite efficient constructs producing the maximally-contained rewriting. Most of them have been developed for conjunctive views and queries in the relational, actually typeless data models (Datalog). In spite of significant progress of query rewriting with views, it remains unclear how to rewrite queries in the typed, object-oriented mediator environment. This paper is an attempt to fill in this gap. The paper embeds conjunctive views and queries into an advanced canonical object model of the mediator [9,11]. The “selection-projection-join” (SPJ) conjunctive query semantics based on type specification calculus [10] is introduced. The paper shows how the existing query rewriting approaches can be extended to be applicable in such object framework. To be specific, the algorithm for rewriting unions of general conjunctive queries [17] has been chosen. The resulting algorithm for the typed environment proposed in the paper exploits the heterogeneous source registration facilities [1] that are based on the refining mapping of the specific source data models into the canonical model of the mediator, resolving ontological differences between mediated and local concepts as well as between structural, behavioral and value conflicts of local and mediated types and classes. Due to the space limit, this paper does not consider various aspects of query rewriting, e.g., such issues as complexity of rewriting, possibility of computing all certain answers to a union query are not discussed: these issues are built on a well known results in the area (e.g., it is known that the inverse-rules algorithm produces

the maximally-contained rewriting in time that is polynomial in the size of the query and the views [8]).
The paper is structured as follows. After brief analysis of the related works, an overview of the basic features of the canonical object model of the subject mediator is given. This overview is focused mostly on the type specification operations of the model that constitute the basis for object query semantics. In Section 4 an object query language oriented on representation of union of conjunctive queries under SPJ set semantics is introduced. Section 5 and 6 provide source registration and query rewriting approach for such query language. Section 7 gives an example of query rewriting in the typed environment. Results are summarized in the conclusion.

2 Related work

The state of the art in the area of answering queries using views ranging from theoretical foundations to algorithm design and implementation has been surveyed in [8]. Additional evaluations of the query rewriting algorithms can be found in the recent papers [7,17] that have not been included into the survey [8]. Inverse rules algorithms are recognized due to their conceptual simplicity, modularity and ability to produce the maximally-contained rewriting in time that is polynomial in the size of the query and the views. Rewriting unions of general conjunctive queries using views [17] compares favorably with existing algorithms, it generalizes the MiniCon [14] and U-join [15] algorithms and is more efficient than the Bucket algorithm. Finding contained rewritings of union queries using general conjunctive queries (when the query and the view constraints both may have built-in predicates) are important properties of the algorithm [17].

Studies of the problem of answering queries using views in the context of querying object-oriented databases [4,5] exploited some semantic information about the class hierarchy as well as syntactic peculiarities of OQL. No concern of object query semantics in typed environment has been reported.

In the paper [6] in different context (logic-based query optimization for object databases) it has been shown how the object schema can be represented in Datalog. Semantic knowledge about the object data model, e.g., class hierarchy information, relationship between objects, as well as semantic knowledge about a particular schema and application domain are expressed as integrity constraints. An OQL object query is represented as a logic query and query optimization is performed in the Datalog representation.

Main contribution of our work is providing an extension of the query rewriting approach using views for the typed subject mediation environment. In contrast with [17], we extend conjunctive queries with object SPJ semantics based on type refinement relationship and type calculus. The paper shows that refinement of the mediator class instance types by the source class instance types is the basic relationship required for establishing query containment in the object environment.

3 Overview of the basic features of the canonical model

In the project [12] for the canonical model of a mediator we choose the SYNTHESIS language [9] that is a hybrid semi-structured/object model [11]. Here only the very basic canonical language features are presented to make the examples demonstrating ideas of the query rewriting readable. It is important to note that the compositional specification calculus considered [10] does not depend on any specific notation or modeling facilities. The canonical model [9] provides support of wide range of data - from untyped data on one end of the range to strictly typed data on another.

Typed data should conform to *abstract data types* (ADT) prescribing behaviour of their instances by means of the type's operations. ADT describes interface of a type whose signature define names and types of its operations. Operation is defined by a predicative specification stating its mixed pre/post conditions. Object type is a subtype of a non-object ADT with an additional operation *self* on its interface providing OIDs. In this paper only typed capabilities of the SYNTHESIS language are exploited. Query rewriting with semi-structured (frame) data is planned to be discussed in the future works.

Sets in the language (alongside with bags, sequences) are considered to be a general mechanism of grouping of ADT values. A class is considered as a subtype of a set type. Due to that these generally different constructs can be used quite uniformly: a class can be used everywhere where a set can be used. For instance, for the query language formulae the starting and resulting data are represented as sets of ADT values (collections) or of objects (classes).

3.1 Type specification operations

Semantics of operations over classes in the canonical model are explained in terms of the compositional specification calculus [10]. The manipulations of the calculus include decomposition of type specifications into consistent fragments, identification of common fragments, composition of identified fragments into more complex type specifications conforming to the resulting types of the SPJ operations. The calculus uses the following concepts and operations.

A signature Σ_T of a type specification $T = \ <V_T, O_T, I_T>$ includes a set of operation symbols O_T indicating operations argument and result types and a set of predicate symbols I_T (for the type invariants) indicating predicate argument types. Conjunction of all invariants in I_T constitutes the type invariant. We model an extension V_T of each type T (a carrier of the type) by a set of proxies representing respective instances of the type.

Definition 1. *Type reduct* A signature reduct R_T of a type T is defined as a subsignature Σ'_T of type signature Σ_T that includes a carrier V_T, a set of symbols of operations $O'_T \subseteq O_T$, a set of symbols of invariants $I'_T \subseteq I_T$.

This definition from the signature level can be easily extended to the specification level so that a type reduct R_T can be considered a *subspecification* (with a signature

Σ_T) of specification of the type *T*. The specification of R_T should be formed so that R_T becomes a supertype of *T*. We assume that only the states admissible for a type remain to be admissible for a reduct of this type (no other reduct states are admissible). Therefore, the carrier of a reduct is assumed to be equal to the carrier of its type.

Definition 2. Type *U* is a *refinement* of type *T* iff

- there exists a one-to-one correspondence *Ops*: $O_T \leftrightarrow O_U$;
- there exists an abstraction function *Abs:* $V_U \rightarrow V_T$ that maps each admissible state of *U* into the respective state of *T*;
- $\forall x \in V_U (I_U(x) \rightarrow I_T (Abs(x)))$;
- for every operation $o \in O_T$ the operation $Ops(o) = o' \in O_U$ is a refinement of *o*. To establish an operation refinement it is required that operation precondition *pre(o)* should imply the precondition *pre(o')* and operation postcondition *post(o')* should imply postcondition *post(o)*.

Based on the notions of reduct and type refinement, a measure of common information between types in the type lattice $\mathcal{T}$ can be established. *Subtyping* is defined similarly to the refinement, but *Ops* becomes an injective mapping.

Definition 3. *Type meet operation*. An operation $T_1 \cap T_2$ produces a type *T* as an 'intersection' of specifications of the operand types. Let $T_1 = <V_{T1}, O_{T1}, I_{T1}>$, $T_2 = <V_{T2}, O_{T2}, I_{T2}>$, then $T = <V_T, O_T, I_T>$ is determined as follows. O_T is produced as an intersection of O_{T1} and O_{T2} formed so that if two methods – one from O_{T1} and another one from O_{T2} – are in a refinement order, then the most abstract one is included in O_T, $I_T = I_{T1} \vee I_{T2}$. *T* is positioned in a type lattice as the most specific supertype of T_1 and T_2 and a direct subtype of all common direct supertypes of the meet argument types.

If one of the types T_1 or T_2 or both of them are non-object types then the result of meet is a non-object type[2]. Otherwise it is an object type. If T_2 (T_1) is a subtype of T_1 (T_2) then T_1 (T_2) is a result of the meet operation.

Definition 4. *Type join operation*. An operation $T_1 \cup T_2$ produces type *T* as 'join' of specifications of the operand types. Let $T_1 = <V_{T1}, O_{T1}, I_{T1}>$, $T_2 = <V_{T2}, O_{T2}, I_{T2}>$, then $T = <V_T, O_T, I_T>$ is determined as follows. O_T is produced as a union of O_{T1} and O_{T2} formed so that if two methods – one from O_{T1} and another one from O_{T2} – are in a refinement order, then the most refined one is included in O_T, $I_T = I_{T1}$ & I_{T2}. *T* is positioned in a type lattice as the least specific subtype of T_1 and T_2 and a direct supertype of all the common direct subtypes of the join argument types.

[2] If a result of type meet or join operation is an object type then an instance of the resulting type includes a *self* value taken from a set of not used OID values.

If one of the types T_1 or T_2 or both of them are object types then the result of join is an object type. Otherwise it is a non-object type. If T_2 (T_1) is a subtype of T_1 (T_2) then T_2 (T_1) is a result of a join operation.

Operations of the compositional calculus form a *type* lattice [10] on the basis of a subtype relation (as a partial order). In the SYNTHESIS language the type composition operations are used to form type expressions that in a simplified form look as follows:

```
<type expression>::= <type term>[<compositional operation><type expression>] |
        (<type expression>)
<compositional operation>::= ∩ | ∪
<type term>::= <type variable> | <type designator> | <function designator>
<type designator>::= <type name> | <attribute name> | <reduct>
<reduct>::= <type name>[<attribute name list>]
<attribute name>::= <identifier> [/<type name>] [:<attribute path expression>]
<attribute path expression>::= <identifier>[.<identifier>]…
```

Reduct *T[b/S]* where *b* is an attribute of type *T* and *S* is a supertype of a type of the attribute *b*, denotes a type with the attribute *b* of type *S*. Reduct *T[a:b.c]* where *a* is an identifier, *b* is an attribute of type *T*, type of the attribute *b* is *S*, *c* is an attribute of *S* and type of the attribute *c* is *U,* denotes a type with the only attribute *a* of type *U*. Type expressions are required to type variables in formulae:

```
<typed variable>::= <variable>/<type expression>
```

3.2 Mediator schema example

For the subject mediator example a Cultural Heritage domain is assumed and a respective mediated schema is provided (table 1, table 2). Attribute types that are not specified in the table should be string, text (*title, content, description, general_Info*) or time (various dates are of this type). Text type is an ADT providing predicates used for specifying basic textual relationships for text retrieval. Time type is an ADT providing temporal predicates. Mentioning text or time ADT in examples means their reducts that will be refined by respective types in the sources. These types require much more space to show how to treat them properly. Therefore in the examples we assume texts to be just strings and dates to be of integer type. In the schema *value* is a function giving an evaluation of a heritage entity cost (+/- marks input/output parameters of a function).

Table 1. Classes of the mediated schema

Class	Subclass of	Class inst. type/ Type	Subtype of
heritage_entity		*Heritage_Entity*	*Entity*
painting	*heritage_entity*	*Painting*	*Heritage_Entity*
sculpture	*heritage_entity*	*Sculpture*	*Heritage_Entity*
antiquities	*heritage_entity*	*Antiquities*	*Heritage_Entity*
museum		*Repository*	
creator		*Creator*	*Person*

Table 2. Types of the mediated schema

Type	Attributes
Entity	*title, date, created_by: Creator, value:{in:function; params:{+e/Entity[title, n/created_by.name],-v/real}}*
Heritage_Entity	*place_of_origin, date_of_origin, content, in_collection: Collection, digital_form:Digital_Entity*
Painting	*dimensions*
Sculpture	*material_medium, exposition_space:{sequence; type_of_element: integer}*
Antiquities	*type_specimen, archaeology*
Repository	*name, place, collections: {set-of: Collection}*
Collection	*name, location. description, in_repository: Repository, contains:{set-of: Heritage_Entity}*
Person	*name, nationality, date_of_birth, date_of_death, residence*
Creator	*culture, general_Info, works: {set-of: Heritage_Entity}*

4 Subset of the SYNTHESIS query language

In the paper a limited subset of the SYNTHESIS query language oriented on representation of union of conjunctive queries under SPJ set semantics is experienced. To specify query formulae a variant of a typed (multisorted) first order predicate logic language is used. Predicates in formulae correspond to collections (such as sets and bags of non-object instances), classes treated as set subtypes with object-valued instances and functions. ADTs of instance types of collections and classes are assumed to be defined. Predicate-class (or predicate-collection) is always a unary predicate (a class or collection atom). In query formulae functional atoms[3] corresponding to functions F syntactically are represented as n-ary predicates $F(X,Y)$ where X is a sequence of terms corresponding to input parameters $i_0, \dots , i_r$ (i_0 is an input parameter typed by an ADT (or its reduct) that includes F as its functional attribute (method); $i_1, \dots , i_r$ – input parameters having arbitrary types ($r \geq 0$)) and Y is a sequence of typed variables having arbitrary types ($s \geq 1$) and corresponding to output parameters $o_1, \dots ,o_s$. For terms the expressions (in particular cases - variables, constants and function designators) can be used. Each term is typed.

[3] State-based and functional attributes are distinguished in type definitions. Functional attributes are taken (similarly to [6]) out of the instance types involved into the formula to show explicitly and plan the required computations.

Definition 6. *SYNTHESIS Conjunctive Query (SCQ, also referred as a rule)* is a query of the form $q(v/T_v)$:- $C_1(v_1/T_{v1})$, ... , $C_n(v_n/T_{vn})$, $F_1(X_1,Y_1)$, ... , $F_m(X_m,Y_m)$, B where $q(v/T_v)$, $C_1(v_1/T_{v1})$, ... , $C_n(v_n/T_{vn})$ are collection (class) atoms, $F_1(X_1,Y_1)$, ... , $F_m(X_m,Y_m)$ are functional atoms, B, called constraint, is a conjunction of predicates over the variables v, v_1, ... , v_n, typed by T_v, T_{v1}, ... , T_{vn} , or output variables $Y_1 \cup Y_2 \cup \ldots \cup Y_m$ of functional atoms. Each atom $C_i(v_i/T_{vi})$ or $F_j(X_j,Y_j)$ ($i = 1, \ldots , n; j = 1, \ldots ,m$) is called a subgoal. The value v structured according to T_v is called the output value of the query. A union query is a finite union of SCQs. Atoms $C_i(v_i/T_{vi})$ may correspond to intensional collections (classes) that should be defined by rules having the form of SCQ[4].

The SPJ set semantics of SCQ body is introduced further[5]. General schema of calculating a resulting collection for a body of SCQ $C_1(v_1/T_{v1})$, ... , $C_n(v_n/T_{vn})$, $F_1(X_1,Y_1)$, ... , $F_m(X_m,Y_m)$, B is as follows. First, Cartesian product of collections in the list is calculated. After that for each instance obtained functional predicates are executed. A method to be executed is determined by a type of the first argument of a functional predicate. A type of the resulting collection of a product is appended with the attributes of the output arguments for each function in SCQ. Each instance of the resulting collection of a product is appended with the values of the output arguments for each function in SCQ calculated for this particular instance. After that those instances of the resulting collections are selected that satisfy B. After that joins of product domains are undertaken. We assume that such joins are executed in the order of appearance of the respective collection predicates in the SCQ body, from the left to the right. The semantics of SCQ conjunctions $C_i(v_i/T_{vi})$, $C_j(v_j/T_{vj})$ are defined by the instance types of the arguments resulting in a type defined by *join* operation of the specifications of the respective argument types. Formal semantics of SCQ are given in Appendix.

The semantics of disjunctions $C_i(v_i/T_{vi}) \lor C_j(v_j/T_{vj})$ requires that for T_{vi} and T_{vj} a resulting type of disjunction is defined by type operation *meet* and the disjunction means a union of C_i and C_j. If the result of *meet* is empty then the disjunction is undefined. Note that in atom $C_i(v_i/T_{vi})$ for T_{vi} any reduct of C_i instance type may be used. This leads to a "projection" semantics of $C_i(v_i/T_{vi})$.

Under such interpretation, SCQ (or a rule) is *safe* if the instance type of the SCQ (rule) head is a supertype of the resulting type of the SCQ (rule) body. Such resulting type may include also attributes equated (explicitly or implicitly) by B to a variable or to a constant. "Implicitly" may mean that such attribute is equated to output argument of a function.

Two SCQs $q_1(v_1/T_{v1})$ and $q_2(v_2/T_{v2})$ are said to be *comparable* if T_{v1} is a subtype of T_{v2}. Let q_1 and q_2 be two comparable queries. q_1 is said *to be contained* in q_2, denoted $q_1 \subseteq q_2$, if for any database instance, all of the answers to q_1 after their transformation to type T_{v2} are answers to q_2.

[4] To make presentation more focused, in the paper everywhere non-recursive SCQs and views are assumed.

[5] For the bag semantics additionally it is required to ensure that the multiplicity of answers to the query are not lost in the views (applying set semantics), and are not increased.

5 Sources registration and inverse rules

During the registration a local source class is described as a view over virtual classes of the mediator having the following general form of SCQ.

$$V(h/T_h) \subseteq P_1(b_1/T_{b1}), \dots , P_k(b_k/T_{bk}), F_1(X_1,Y_1), \dots , F_r(X_r,Y_r), B$$

Here V is a source class, $P_1, \dots , P_k$ are classes of the mediator schema[6], $F_{b1}, \dots , F_{br}$ are functions of the mediator schema, B is a constraint imposed by the view body. This SCQ should be safe. The safety property is established during the registration process. Due to that one-to-one correspondence between attributes of a reduct of the resulting instance type of a view body (mediator) and the instance type of a view head (source) is established. On source registration [1] this is done so that a reduct of the view body instance type *is refined* by the concretizing type T_h designed above the source. Since the open world assumption is applied, each class instance in a view may contain only part of the answers computed to the corresponding query (view body) on the mediator level. To emphasize such incompleteness, a symbol $\subseteq$ is used to interconnect a head of the view with its body.

For the local sources of our Cultural Heritage domain example few schemas are assumed for Louvre and Uffizi museum Web sites. Several view definitions for these sources registered at the mediator follow. In the views it is assumed that reducts of their instance types refine reducts of the respective mediator classes instance types or their compositions. Details on that can be found in [1]. The same is assumed for attribute types having the same names in a view and in the mediator.

Uffizi site views

canvas(p/Canvas[title, name, culture, place_of_origin, r_name]) $\subseteq$
painting(p/Painting[title, name: created_by.name, place_of_origin, date_of_origin, r_name: in_collection.in_repository.name]), creator(c/Creator[name, culture]), r_name = 'Uffizi', date_of_origin >= 1550, date_of_origin < 1700

artist (a/Artist[name, general_Info, works]) $\subseteq$ *creator(a/Creator[name, general_Info, works/{set-of:Painting}])*

Louvre site views

workP(p/Work[title, author, place_of_origin, date_of_origin, in_rep]) $\subseteq$
painting(p/Painting[title, author: created_by.name, place_of_origin, date_of_origin, in_rep: in_collection.in_repository.name]), in_rep = 'Louvre'

workS(p/Work[title, author, place_of_origin, date_of_origin, in_rep]) $\subseteq$
sculpture(p/Sculpture[title, author: created_by.name, place_of_origin, date_of_origin, in_rep: in_collection.in_repository.name]), in_rep = 'Louvre'

[6] These atoms may also correspond to intensional classes defined by rules in the mediator schema.

To produce inverse rules out of the mediator view definitions as above, first, replace in the view each not contained in T_h attribute from $T_{b1}, \dots ,T_{bk}$ with a distinct Skolem function of h/T_h producing output value of the type of the respective attribute. Such replacing means substitution of the attribute *get* function in a type with a method defined as a Skolem function that can not be expressed in terms of a local source. In the text Skolemized attributes are marked with #. All Skolemized attributes are added to the type T_h. Such Skolemizing mapping of the view is denoted as ρ. After the Skolemizing mapping, inverse rules for the mediator classes in the view body are produced as $\rho(P_i(b_i/T_{bi}) \leftarrow V(h/T_h))$ (for $i = 1, \dots , k$)[7]. It is assumed that types T_{bi} and T_h are defined as $T_{bi}\ [a_1/T_1:t_1, \dots , a_n/T_n:t_n]$ and $T_h[a_1/S_1, \dots , a_m/S_m]$ so that T_{bi} is a supertype of T_h and T_i is a supertype of S_i, $i = 1, \dots , n$.

For the mediator functions being type methods the inverse rules look like $\rho([m/T_m.]F_{bj}(X_{bj},Y_{bj}) \leftarrow [s/T_s.]F_{hl}(X_{hl},Y_{hl}))$, for $j = 1, \dots ,r$, here F_{bj} and F_{hl} are methods of T_m and T_s such that type of function of F_{hl} refines type of function of F_{bj}. Types T_m and T_s (defined for the mediator and a source respectively) may be given implicitely in the view definition and (or) deduced from the registration information. Functional inverse rules are obtained during the source registration at the mediator, not from the view body.

$\rho(B_b)$ will be called the *inferred constraint* of the view predicate $V(h/T_h)$. The inverse rules generated from different views must use different Skolem functions (indexing of # will denote this).

Inverse rules for canvas view of Uffizi:

painting(p/Painting[title, name: created_by.name, place_of_origin, $\#_1$date_of_origin, r_name: in_collection.in_repository.name]) $\leftarrow$ *canvas(p/ Canvas[title, name, culture, place_of_origin, $\#_1$date_of_origin, r_name])*

creator(c/ Creator[name, culture]) $\leftarrow$ *canvas(p/ Canvas[title, name, culture, place_of_origin, r_name])*

The inferred constraint for *canvas(p/Canvas[title, name, culture, place_of_origin, $\#_1$date_of_origin, r_name])*:
r_name = 'Uffizi', $\#_1$date_of_origin >= 1550, $\#_1$date_of_origin < 1700

During Uffizi registration a functional inverse rule is registered as

value(h/Entity[title, name: created_by.name], v/real) $\leftarrow$ *amount(h/Entity[title, name: created_by.name], v/real)*

[7] If T_{bi} is an object type then common reduct of T_{bi} and T_h should be such that *self* attribute of T_{bi} has an interpretation in T_h, otherwise *self* is to be replaced with a Skolem function generating OIDs.

Inverse rules for workP and workS views of Louvre:

painting(p/Painting[title, author: created_by.name, place_of_origin, date_of_origin, in_rep: in_collection.in_repository.name]) $\leftarrow$ *workP(p/Work[title, author, place_of_origin, date_of_origin, in_rep])*

The inferred constraint for *workP*: *in_rep = 'Louvre'*

To save space, similar rule for sculpture is not shown here. During Louvre registration a functional inverse rule is registered as

value(h/Entity[title, name: created_by.name], v/real) $\leftarrow$ *amount(h/Entity[title, name: created_by.name], v/real)*

Given a union query Q_m defined over the mediator classes, collections and functions, our task is to find a query Q_v defined solely over the view classes, collections and functions such that, for any mediator database instance, all of the answers to Q_v computed using any view instance are correct answers to Q_m. Q_v should be a *contained rewriting* of Q_m. There may be many different rewritings of a query.

6 Query rewriting

Let Q_u be a union mediator query to be rewritten. Without loss of generality, all the SCQs in Q_u are assumed to be comparable. Similarly to [17], the method for rewriting Q_u consists of two steps. In the first step, we generate a set of *candidate formulae* (candidates for short) which may or may not be rewritings. These candidates are generated separately for every SCQ in Q_u. In the second step, all these candidates are checked to see whether correct rewritings can be obtained. A set I of compact inverse rules is assumed to be obtained for various sources as a result of their registration at the mediator [1].

For each SCQ $q(v/T_v)$:- $C_1(v_1/T_{v1}), \ldots, C_n(v_n/T_{vn}), F_1(X_1,Y_1), \ldots, F_m(X_m,Y_m), B$ in Q_u denoted as Q do the following.

For each subgoal $C_i(v_i/T_{vi})$ or $F_j(X_j,Y_j)$ of Q find inverse rule $r \in I$ with the head $P_l(b_l/T_{bl})$ or $F_{bo}(X_{bo},Y_{bo})$ such that $C_i = P_l$ (or P_l is a name of any transitive subclass of C_i) and T_{bl} is a subtype of T_{vi} (or $F_j = F_{bo}$ and F_{bo} function type is a refinement of F_j type). Further such discovery is called *subgoal unification*.

A *destination* of Q is a sequence D of atoms $P_1(b_1/T_{b1}), \ldots, P_n(b_n/T_{bn}), F_{b1}(X_{b1},Y_{b1}), \ldots, F_{bm}(X_{bm},Y_{bm})$ obtained as a result of the query subgoals unification with the heads of inverse rules from I. Several destinations can be produced as various combinations of SCQ subgoals unifications found. Additionally each destination should conform to the following constraints:

1. There is no j such that a constant in X_j of $F_j(X_j,Y_j)$ ($j=1, \ldots, m$) of Q corresponds to a different constant in the same argument of the respective functional subgoal of destination $F_{bj}(X_{bj},Y_{bj})$.

2. No two occurrences of the same variable or of the same function in F_j ($j=1, \dots , m$) of Q correspond to two different constants in F_{bj} ($j=1, \dots , m$) of D, and no two occurrences of the same variable or of the same function in the same head of a rule correspond to two different constants in Q.

Once a destination D of Q is found, we can use it to construct a candidate formula as follows. For each atom $P_i(b_i/T_{bi})$ or $F_{bj}(X_{bj}, Y_{bj})$ in D (supposing it is a head of the inverse rule $P_i(b_i/T_{bi}) \leftarrow V_i(h_i/T_{hi})$ resp. $F_{bj}(X_{bj}, Y_{bj}) \leftarrow F_{hj}(X_{hj}, Y_{hj})$) do the following (if there are rules that have the same head but different bodies, then choose one of them in turn to generate different candidates).

Establish a mapping ϕ_i of attributes and variables in the atom $P_i(b_i/T_{bi})$ of D and in the associated atom $V(h_i/T_{hi})$ to the attributes and variables of the respective atom $C_i(v_i/T_{vi})$ of Q. For each variable z in T_{hi} of $V(h_i/T_{hi})$ which does not appear in $P_i(b_i/T_{bi})$ as a free variable, let ϕ_i map z to a distinct new variable not occurring in Q or any other view atom $\phi_j(V_j(h_j/T_{hj}))$, ($i \neq j$). Free variables in the atom $P_i(b_i/T_{bi})$ of D and its associated atom $V(h_i/T_{hi})$ are mapped to the respective atom of Q as follows. For the atom $P_i(b_i/T_{bi})$ of D and the associated atom $V(h_i/T_{hi})$ the mappings of b_i and h_i to v_i are added to ϕ_i. For all T_{hi} attributes do the following. If an attribute a does not belong to T_{vi} then add to ϕ_i the mapping of a to an empty attribute (i.e., remove the attribute). If an attribute a belongs to T_{vi} but it has the form a/R where R is a supertype of a type of the attribute a, then add to ϕ_i the mapping of a to a/R or to $a/R{:}\#a$ if a is a Skolem attribute. If an attribute a of the type T_{vi} contains an attribute of the form $b/R{:}t.s$ and the type T_{bi} contains an attribute of the form $a/T{:}t$ (where t and s are attribute path expressions, R is a supertype of T and T is a supertype of a type of a) then add to ϕ_i the mapping of a to $b/R{:}a.s$ or to $a/R{:}\#a$ if a is a Skolem attribute. Similarly we build a mapping ϕ_j of attributes and variables in the atom $F_{bj}(X_{bj}, Y_{bj})$ of D and in the associated atom $F_{hj}(X_{hj}, Y_{hj})$ to the attributes and variables of the respective atom $F_j(X_j, Y_j)$ of Q.

For each destination and variable mappings defined, construct a formula Φ.

$$\phi_1(P_1(b_1/T_{b1})), \dots , \phi_n(P_n(b_n/T_{bn})), \phi_{n+1}(F_{b1}(X_{b1}, Y_{b1})), \dots , \phi_{n+m}(F_{bm}(X_{bm}, Y_{bm})) \qquad (\Phi)$$

Construct the mapping δ of a constraint of Q to a constraint in Φ. Let $S_q = (a_1, a_2, \dots , a_k)$ and $S_f = (c_1, c_2, \dots , c_k)$ be the sequences of function arguments in Q and Φ respectively. The mapping δ is constructed as follows.

Initially, an associated equality of the constraint in Φ, $E_\delta = True$. For $i = 1$ to m:

1. If a_i is a constant α or a function that results in α, but c_i is a variable y, then let $E_\delta = E_\delta \wedge (y = \alpha)$. α should be of y type or any of its subtypes.
2. If a_i is a variable x, and x appears the first time in position i, then let δ map x to c_i. If x appears again in a later position $j > i$ of S_q, and $c_i \neq c_j$, then let $E_\delta = E_\delta \wedge (c_i = c_j)$. a_i, c_i types and a_j, c_j types are assumed to be the same or in a subtyping order. We shall get SCQ:

$$q(v/T_v)\text{:- } \phi_1(P_1(b_1/T_{b1})), \dots, \phi_n(P_n(b_n/T_{bn})), \phi_{n+1}(F_{b1}(X_{b1},Y_{b1})), \dots, \phi_{n+m}(F_{bm}(X_{bm},Y_{bm})), \delta(B), E_\delta \qquad (\Phi_1)$$

Replace heads of the inverse rules in the above SCQ with the rules bodies to get the formula

$$q(v/T_v)\text{:- } \phi_1(V_1(h_1/T_{h1})), \dots, \phi_n(V_n(h_n/T_{hn})), \phi_{n+1}(F_{h1}(X_{h1},Y_{h1})), \dots, \phi_{n+m}(F_{hm}(X_{hm},Y_{hm})), \delta(B), E_\delta \qquad (\Phi_2)$$

If the constraint $\delta(B) \wedge E_\delta$ and the inferred constraints of the view atoms in the candidate formula *are consistent* and there are no Skolem functions in the candidate of Q then the formula is a rewriting (remove duplicate atoms if necessary). If there are Skolem functions in $\delta(B) \wedge E_\delta$, then the candidate formula is not a rewriting because the values of Skolem functions in $\delta(B) \wedge E_\delta$ can not be determined. Note that $x=y$ in the constraint for terms x and y typed with ADTs T and S is recursively expanded as $x.a_1=y.a_1$ & ... & $x.a_n=y.a_n$ where $a_{1,\ \dots\ ,}\ a_n$ are common attributes of T and S.

Containment property of the candidate formulae. The candidate formula (Φ_2) has the following property. If we replace each view atom with the corresponding Skolemized view body and treat the Skolem functions as variables, then we will get a safe SCQ Q' (the expansion of the candidate formula (Φ_2)) which is contained in Q. This is because (Φ_1) is a safe SCQ which is equivalent to $\delta(Q)$, and all subgoals and built-in predicates of (Φ_1) are in the body of Q' (this is a containment mapping). We constructed Φ_2 so that for any collection (class) subgoal pair in Φ_2 and Q an instance type of a subgoal of Φ_2 is a refinement of the instance type of the respective subgoal of Q, for any functional subgoal pair in Φ_2 and Q a type of function of a subgoal of Φ_2 is a refinement of type of function of the respective subgoal of Q. In some cases it is possible to obtain rewritings from the candidate formulae eliminating Skolem functions [17]. If the inferred constraints of the view atoms imply the constraints involving Skolem functions in the candidate formula, then we can remove those constraints directly.

Consistency checking

Main consistency check during the rewriting consists in testing that constraint $\delta(B) \wedge E_\delta$ together with the inferred constraints of the view atoms in a candidate formula are consistent. Here we define how it can be done for the arithmetic constraints following complete algorithm for checking implications of arithmetic predicates [16].

1. Assuming that in SCQ we can apply only arithmetic predicates, form $Arith = \delta(C) \wedge E_\delta \wedge <$inferred constraints of the view atoms in a candidate formula $\Phi>$.
2. For a candidate formula Φ it is required to show that there exist correct substitutions of type attributes and function arguments[8] in Φ satisfying $Arith$.

[8] It follows that an ability to compute functions during the consistency check to form admissible combination of input – output argument values is required.

7 Query rewriting example in a subject mediator

Rewrite the following query to the Cultural Heritage domain mediator:

valuable_Italian_heritage_entities(h/Heritage_Entity_Valued[title, c_name, r_name, v]) :- heritage_entity(h/Heritage_Entity[title, c_name:created_by.name, place_of_origin, date_of_origin, r_name: in_collection.in_repository.name]), value(h/ Heritage_Entity [title, name: c_name], v/real), v >= 200000, date_of_origin >= 1500, date_of_origin < 1750, place_of_origin = 'Italy'

Destinations obtained

For Uffizi site *heritage_entity* subgoal of a query unifies with *painting* as a *heritage_entity* subclass. The first destination is obtained as:

painting(p/Painting[title, name: created_by.name, place_of_origin, $\#_1$*date_of_origin, r_name: in_collection.in_repository.name]), value(h/Entity[title,name: created_by.name], v/real)*

ϕ mapping for the destination (only different name mappings are shown):

ϕ_1 mapping	*p → h,* $\#_1$*date_of_origin → date_of_origin:*$\#_1$*date_of_origin, r_name → c_name: r_name*
ϕ_2 mapping	*name: created_by_name → name:c_name*

For the query constraint δ is an identity mapping and E_δ*=true.* Applying

ϕ_1*(canvas(p/Canvas[title, name, culture, place_of_origin, r_name])),*
ϕ_2*(amount(h/Entity[title, name: created_by.name], v/real)), v >= 200000, date_of_origin >= 1500, date_of_origin < 1750, place_of_origin = 'Italy'*

we get the candidate formula

valuable_Italian_heritage_entities(h/Heritage_Entity_Valued[title, c_name, r_name, v]) :- canvas(h/Canvas[title, name, culture, place_of_origin, date_of_origin: $\#_1$*date_of_origin, c_name: r_name]), amount(h/Entity[title, name: c_name], v/real), v >= 200000, date_of_origin >= 1500, date_of_origin < 1750, place_of_origin = 'Italy'*

For Louvre the *heritage_entity* subgoal of a query unifies with *painting, sculpture* as *heritage_entity* subclasses. Only destination formed for *painting* is shown here. This second destiniation is obtained as:

painting(p/Painting[title, author: created_by.name, place_of_origin, date_of_origin, in_rep: in_collection.in_repository.name]), value(h/Entity[title, name: created_by.name], v/real)

ϕ mapping for the destination:

ϕ_1 mapping	*author → c_name:author, in_rep → r_name:in_rep*

ϕ_2 mapping	*name:created_by.name* $\rightarrow$ *name:c_name*

Again, δ is an identity mapping and E_δ=*true*. Finally we get the second candidate formula

valuable_Italian_heritage_entities(h/Heritage_Entity_Valued[title, c_name, r_name, v]) :-
workP(h/Work[title, c_name: author, place_of_origin, date_of_origin, r_name: in_rep]),
amount(h/[title, name: c_name], v/real), v >= 200000, date_of_origin >= 1500,
date_of_origin < 1750, place_of_origin = 'Italy'

Obtaining rewritings from candidate formulae
To retrieve a rewriting we eliminate Skolem functions from the first candidate formula.. Note that the inferred constraint for *canvas(h/Canvas[title, c_name: name, culture, place_of_origin, date_of_origin:* $\#_1$*date_of_origin, r_name])* that looks as *r_name = 'Uffizi',* $\#_1$*date_of_origin >= 1550,* $\#_1$*date_of_origin < 1700* implies *date_of_origin >= 1500, date_of_origin < 1750* for Uffizi. Due to that Skolem functions can be eliminated from this candidate formula and after the consistency check we get the following rewriting:

valuable_Italian_heritage_entities(h/Heritage_Entity_Valued[title, c_name, r_name, v]) :-
canvas(h/ Canvas[title, c_name: name, culture, place_of_origin, r_name]), amount(h/[title, name: c_name], v/real), v >= 200000, place_of_origin = 'Italy'

The second candidate formula is a correct rewriting without any transformation.

Conclusion

The paper presents a query rewriting method for the heterogeneous information integration infrastructure formed by a subject mediator environment. LAV approach treating schemas exported by sources as materialized views over virtual classes of the mediator is considered as the basis for the subject mediation infrastructure. Main contribution of this work consists in providing an extension of the query rewriting approach using views for the typed environment of subject mediators. Conjunctive views and queries are considered in frame of an advanced canonical object model of the mediator. The "selection-projection-join" (SPJ) conjunctive query semantics based on type specification calculus has been introduced. The paper shows how the existing query rewriting approaches can be extended to be applicable in such typed framework. The paper demonstrates that refinement of the mediator class instance types by the source class instance types is the basic relationship required for query containment in the typed environment to be established. The approach presented is under implementation for the subject mediator prototype [1]. This implementation creates also a platform for providing various object query languages (e.g., a suitable subset of OQL (ODMG) or SQL:1999) for the mediator interface. Such languages can be implemented by their mapping into the canonical model of the mediator.

In a separate paper it is planned to show how an optimized execution plan for the rewritten query is constructed under various limitations of the source capabilities. Future plans include also extension of the query rewriting algorithm for frame-based semi-structured (XML-oriented) queries as well as investigations for queries (views) with negation and recursion.

References

1. D.O. Briukhov, L.A. Kalinichenko, N.A. Skvortsov. Information sources registration at a subject mediator as compositional development. In Proc. of the East European Conference on "Advances in Databases and Information Systems", Lithuania, Vilnius, Springer, LNCS No. 2151, 2001
2. O.M. Duschka, M.R. Genesereth. Query planning in infomaster. In Proc. of the ACM Symposium on Applied Computing, San Jose, CA, 1997
3. O.Duschka, M. Genesereth, A. Levy. Recursive query plans for data integration. Journal of Logic Programming, special issue on Logic Based Heterogeneous Information Systems, 43(1): 49-73, 2000
4. D. Florescu, L. Raschid, P. Valduriez. Answering queries using OQL view expressions. In Workshop on Materialized Views, Montreal, Canada, 1996
5. D. Florescu. Search spaces for object-oriented query optimization. PhD thesis, University of Paris VI, France, 1996
6. J. Grant, J. Gryz, J. Minker, L. Raschid. Semantic query optimization for object databases. In Proc. of the 13th International Conference on Data Engineering (ICDE'97), p.p. 444 – 454, April 1997
7. J. Grant, J. Minker. A logic-based approach to data integration. Theory and Practice of Logic Programming, Vol 2(3), May 2002, 293-321
8. A.Y. Halevy. Answering queries using views: a survey. VLDB Journal, 10(4): 270 – 294, 2001
9. L. A. Kalinichenko. SYNTHESIS: the language for description, design and programming of the heterogeneous interoperable information resource environment. Institute of Informatics Problems, Russian Academy of Sciences, Moscow, 1995
10. L. A. Kalinichenko. Compositional Specification Calculus for Information Systems Development. In Proc. of the East European Conference on Advances in Databases and Information Systems, Maribor, Slovenia, September 1999, Springer Verlag, LNCS No 1691
11. L. A. Kalinichenko. Integration of heterogeneous semistructured data models in the canonical one. In Proc. of the First Russian National Conference on "Digital Libraries: Advanced Methods and Technologies, Digital Collections", Saint-Petersburg, October 1999
12. L.A. Kalinichenko, D.O. Briukhov, N.A. Skvortsov, V.N. Zakharov. Infrastructure of the subject mediating environment aiming at semantic interoperability of heterogeneous digital library collections In Proc. of the Second Russian National Conference on "Digital Libraries: Advanced Methods and Technologies, Digital Collections", Protvino, October 2000
13. A.Y. Levy, A. Rajaraman. J.J. Ordille. Querying Heterogeneous Information Sources Using Source Descriptions. In Proc. of the Int. Conf. on Very Large Data Bases (VLDB), Bombay, India, 1996
14. R.Pottinger, A.Levy. A scalable algorithm for answering queries using views. In Proc. of the Int. Conf. on Very Large Data Bases (VLDB), Cairo, Egypt, 2000
15. X. Qian. Query folding. In Proc. of Int. Conf. on Data Engineering (ICDE), p.p. 48 – 55, New Orleans, LA, 1996

16. M. Staudt, Kai von Thadden. Subsumption checking in knowledge bases. Technical report 95-11, Aachener Informatik – Berichte, RWTH Aachen
17. J. Wang, M.Maher, R. Topor. Rewriting Unions of General Conjunctive Queries Using Views. In Proc. of the 8th International Conference on Extending Database Technology, EDBT'02, Prague, Czech Republic, March 2002

Appendix. Formal semantics of SYNTHESIS Conjunctive Query

Semantics of SCQ ($q(v/T_v)$:- $C_1(v_1/T_{v1})$, ... , $C_n(v_n/T_{vn})$, $F_1(X_1,Y_1)$, ... , $F_m(X_m,Y_m)$, B where $q(v/T_v)$, $C_1(v_1/T_{v1})$, ... , $C_n(v_n/T_{vn})$ are collection (class) atoms, $F_1(X_1,Y_1)$, ... , $F_m(X_m,Y_m)$ are functional atoms, B is a conjunction of predicates over the variables v, v_1, ... , v_n) are given by a semantic function $s[\cdot]$ constructing a result set of SCQ body. $s[\cdot]$ is defined recursively starting with the semantics of collection atoms. Collection C_i is considered as a set of values of type T_{vi}. Any value of type T_{vi} is an element of the extent V_{Tvi} of type T_{vi} . Thus a result set $s[C_n(v_i/T_{vi})]$ of collection atom $C_n(v_i/T_{vi})$ is a subset of the extent V_{Tvi}.

The first stage of constructing of the result set of the SCQ body is as follows. Construct a Cartesian product of sets c_i=$s[C_n(v_i/T_{vi})]$, append elements corresponding to the values of output parameters of functions $F_i(X_i,Y_i)$ to the tuples of the product and select all the tuples satisfying predicate B. Semantic function $ccp[\cdot]$ (conditional Cartesian product) is provided for that:

$ccp[C_1(v_1/T_{v1}), \dots , C_n(v_n/T_{vn}), F_1(X_1,Y_1), \dots , F_m(X_m,Y_m), B] =$
$\{ v_1,\dots, v_n, \zeta_1,\dots, \zeta_m \mid$
$(v_1,\dots, v_n) \in c_1\times\dots\times c_n \wedge$
$\zeta_1\in V_{R1} \wedge \mathcal{F}_1 \wedge \dots \wedge \zeta_m\in V_{Rm} \wedge \mathcal{F}_m \wedge$
$B\{y_1^{\,1} \to \zeta_1.y_1^{\,1},\dots, y_1^{\beta 1} \to \zeta_1.y_1^{\beta 1},\dots, y_m^{\,1} \to \zeta_m.y_m^{\,1},\dots, y_1^{\beta m} \to \zeta_m.y_m^{\beta m}\}$
$\}$

$\mathcal{F}_i$ is a formula defining values of output parameters of F_i in a tuple. To define formally what $\mathcal{F}_i$ is, it is required to make the following assumptions. Let X_i and Y_i be

$X_i= v_{n\gamma i}, x_i^{\,1},\dots, x_i^{\alpha i}$
$Y_i = y_i^{\,1},\dots, y_i^{\beta i}$

Let R_i be a type of the structure of the output parameters of the method F_i.

```
{ Ri; in: type; yi1: Wi1;… yiβi: Wiβi; }
```

Let method F_i has input parameters $a_i^{\,1}/U_i^{\,1},\dots, a_i^{\alpha i}/U_i^{\alpha i}$, output parameters $b_i^{\,1}/W_i^{\,1},\dots, b_i^{\beta i}/W_i^{\beta i}$ and predicative specification f.

Let $Q_1,\dots, Q_{\eta i}$ be all subtypes of the type of the variable $v_{n\gamma i}$ – type $T_{vn\gamma i}$. Let $f_1,\dots, f_{\eta i}$ be predicative specifications of the method F_i for the types $Q_1,\dots, Q_{\eta i}$ respectively. Then formula $\mathcal{F}_i$ (taking into consideration a polymorphism of the method F_i) looks as follows.

$\mathcal{F}_i \risingdotseq$
$v_{n\gamma i} \in V_{Q1} \in f_1\{this\to v_{n\gamma i}, a_i^{\,1}\to x_i^{\,1},\dots, a_i^{\alpha i}\to x_i^{\alpha i}, b_i^{\,1}\to\zeta_i.y_i^{\,1},\dots, b_i^{\beta i}\to\zeta_i.y_i^{\beta i}\} \wedge \dots \wedge$
$v_{n\gamma i} \in V_{Q\eta i} \in f_{\eta i}\{this\to v_{n\gamma i}, a_i^{\,1}\to x_i^{\,1},\dots, a_i^{\alpha i}\to x_i^{\alpha i}, b_i^{\,1}\to\zeta_i.y_i^{\,1},\dots, b_i^{\beta i}\to\zeta_i.y_i^{\beta i}\}$

A notation $f\{a\to t\}$ where f is a formula, a is a variable of f, t is a term, means the formula f with a substituted by t.

The second stage of the construction of the result set is a calculation of joins of product domains. The calculation of a single join is performed by semantic function *sjoin*. It takes a set of *r*-tuples *s* with types of elements $T_1, \ldots, T_r$ and produces a set of *(r-1)*- tuples with types of elements $T_1 \cup T_2, T_3, \ldots, T_r$.

$$sjoin(t) = \{ \mu_1, \ldots, \mu_{r-1} \mid \exists (\lambda_1, \ldots, \lambda_r) \in t, v \in V_{T1 \cup T2} (v =_{T1} \lambda_1 \wedge v =_{T2} \lambda_2 \wedge \mu_1 = v \wedge \mu_2 = \lambda_3 \wedge \ldots \wedge \mu_{r-1} = \lambda_r) \}$$

For every tuple from *t* the function *sjoin* "glues" first two elements $\lambda_1 \in V_{T1}$, $\lambda_2 \in V_{T2}$ of the tuple into one element $\mu_1 \in V_{T1 \cup T2}$. As value of type T_1, μ_1 has all the attributes of the type T_1 and values of these attributes are the same as values of respective attributes of λ_1. As value of type T_2, μ_1 has all the attributes of the type T_2 and values of these attributes are the same as values of respective attributes of λ_2. Equality of values of attributes is expressed by the following notation.

$$v =_T w \Leftrightarrow v.d_1 =_{Q1} w.\ d_1 \wedge \ldots \wedge v.d_g =_{Qg} w.\ d_g$$

Type *T* here has attributes $d_1, \ldots, d_g$ of types $Q_1, \ldots, Q_g$ respectively.
To perform all joins for the product

$$ccp[C_1(v_1/T_{v1}), \ldots, C_n(v_n/T_{vn}), F_1(X_1,Y_1), \ldots, F_m(X_m,Y_m), B]$$

it is required to apply *sjoin* function *n+m-1* times.

In case when all types $T_{v1}, \ldots, T_{vn}$ are nonobject types, the type of the result set is nonobject and the semantic function *s* provided for producing the result set of SCQ right-hand part $C_1(v_1/T_{v1}), \ldots, C_n(v_n/T_{vn}), F_1(X_1,Y_1), \ldots, F_m(X_m,Y_m), B$ is defined as follows.

$$s[C_1(v_1/T_{v1}), \ldots, C_n(v_n/T_{vn}), F_1(X_1,Y_1), \ldots, F_m(X_m,Y_{m1}), B] =$$
$$\underbrace{sjoin(sjoin(\ldots sjoin}_{n+m-1 \text{ times}}(ccp[C_1(v_1/T_{v1}), \ldots, C_n(v_n/T_{vn}), F_1(X_1,Y_1), \ldots, F_m(X_m,Y_{m1}), B]) \ldots))$$

In case when at least one type of $T_{v1}, \ldots, T_{vn}$ is an object type, the type of the result set is object and the semantic function *s* is defined as follows.

$$s[C_1(v_1/T_{v1}), \ldots, C_n(v_n/T_{vn}), F_1(X_1,Y_1), \ldots, F_m(X_m,Y_{m1}), B] =$$
$$objectify(\underbrace{sjoin(sjoin(\ldots sjoin}_{n+m-1 \text{ times}}(ccp[C_1(v_1/T_{v1}), \ldots, C_n(v_n/T_{vn}), F_1(X_1,Y_1), \ldots, F_m(X_m,Y_{m1}), B]) \ldots)))$$

Semantic function *objectify* here converts a collection of nonobject values into a collection of objects. This is done by adding an attribute *self* obtaining some new unique identifier to each value of the nonobject collection.