Containment and Minimization of RDF/S Query Patterns

Giorgos Serfiotis[1,2], Ioanna Koffina[1,2], Vassilis Christophides[1,2], and Val Tannen[3]

[1] Institute of Computer Science, Foundation for Research and Technology – Hellas,
P.O. Box 1385, 71110 Heraklio, Greece
{koffina, christop}@ics.forth.gr
[2] Department of Computer Science, University of Crete,
P.O. Box 2208, 71110 Heraklio, Greece
serfioti@csd.uoc.gr
[3] Department of Computer and Information Science, University of Pennsylvania,
200 South 33rd Street, Philadelphia, Pennsylvania
val@cis.upenn.edu

Abstract. Semantic query optimization (SQO) has been proved to be quite useful in various applications (e.g., data integration, graphical query generators, caching, etc.) and has been extensively studied for relational, deductive, object, and XML databases. However, less attention to SQO has been devoted in the context of the Semantic Web. In this paper, we present sound and complete algorithms for the containment and minimization of RDF/S query patterns. More precisely, we consider two widely used RDF/S query fragments supporting pattern matching at the data, but also, at the schema level. To this end, we advocate a logic framework for capturing the RDF/S data model and semantics and we employ well-established techniques proposed in the relational context, in particular, the Chase and Backchase algorithms.

1 Introduction

Semantic query optimization (SQO) is the process of increasing the potential for an efficient evaluation of queries by using intentional information about the contents of a database. The essential idea is to use knowledge (e.g., under the form of integrity constraints) about the data to reformulate a query into a more efficient but semantically equivalent one. SQO has been proved to be quite useful in various applications. For example, when integrating information sources, the composition of user queries with publishing views that establish mappings between data sources, often results in redundant queries. Moreover, when queries are produced automatically – e.g., from graphical query generators in portals – and not from humans, they are very likely to be redundant. Redundancy can be eliminated using query minimization techniques. Finally, in order to exploit cached query results we need to be able to identify query containment. In this way queries can be (partially or completely) answered, thus, avoiding costly access to remote data sources.

SQO has been extensively studied in the context of relational [19], deductive [4][6][16], object [11] [8] and, recently, XML databases [9][10][18][24]. However, less attention to SQO has been devoted in the context of the Semantic Web. In [1] the authors propose a graphical interface that produces on-the-fly minimal RQL queries while the user navigates through an RDF/S schema. The key idea to query minimization is that when navigating through hierarchies of classes (properties), the path expressions get refined by taking into account the subclasses (subproperties) currently visited. In [23] the author proposes a graph-based approach for identifying RDF/S queries that are subsumed by (i.e., contained in) queries whose results are already cached. If a query *A*, issued on an RDF/S description base *DB*, subsumes a query *B*, then query *B* needs not be executed on *DB*; it can be instead evaluated on the cached results of query *A*. However, both approaches consider only a limited fragment of RDF/S queries featuring pattern matching against data graphs (i.e., similar to relational queries).

In this paper, we study SQO for more expressive fragments of patterns supported by declarative RDF/S query languages. More precisely, we consider two fragments of increasing expressiveness allowing complex pattern matching at the data, but also, at the schema level. The main contribution of this work is to present sound and complete algorithms for both checking the containment of RDF/S query fragments and minimizing them, which generalize previous results [9] for unions of conjunctive queries under disjunctive embedded dependencies. The rest of the paper is organized as follows: Section 2 introduces the logic framework that allows us to reduce the containment and minimization problems to the relational equivalent. Section 3 presents the two expressive RDF/S query fragments for which our algorithms apply. Section 4 and 5 describe the proposed containment and minimization algorithms, which are based on the Chase and Backchase (C&B) algorithms as were introduced in [8] and extended in [9]. Our conclusions, as well as some challenges for future work, are given in Section 6.

2 From RDF/S to SWLF

In this section we will present our logic framework, termed *Semantic Web Logic Framework* (SWLF), consisting of six first-order logic (FOL) predicates for capturing RDF/S schemas and description bases, as well, as a set of appropriate FOL constraints under the form of *disjunctive embedded dependencies* (DEDs).

Table 1. FOL predicates for RDF/S

Predicate	Type
CLASS	Set <name: Class>
PROP	Set <domain: Class, name: Property, range: Class>
C_SUB	Set <subC: Class, class: Class>
P_SUB	Set <subP: Property, prop: Property>
C_EXT	Set <class: Class, inst: Resource>
P_EXT	Set <subject: Resource, prop: Property, object: Resource>

Definition 1. RDF/S schemas and description bases are represented by a set of FOL predicates (relations), namely *R={CLASS, PROP, C_SUB, P_SUB, C_EXT, P_EXT}*. Each predicate R_i has a set of attributes A_i, as shown in **Table 1**, whose domains have one of the following types T_i in *T*, *T={Class, Property, Resource}*.

The meaning of these predicates is given below:

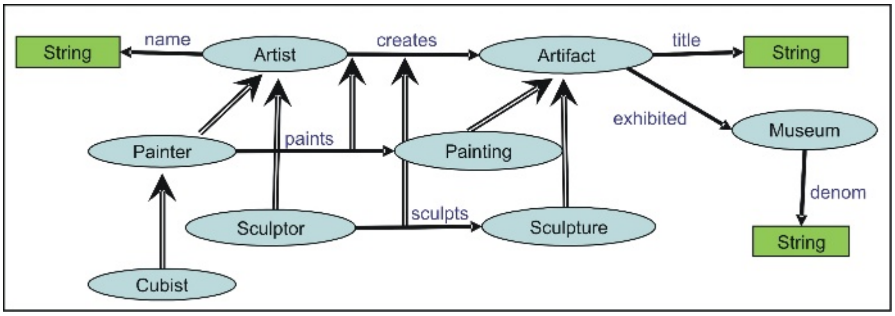

Fig. 1. An RDF/S schema

- CLASS(c) iff *c* is a class.
- PROP(a, p, b) iff *p* is a property having class *a* as domain and class *b* as range.
- C_SUB(c, a) iff class *c* is a (direct or not) subclass of class *a*.
- P_SUB(q, p) iff property *q* is a (direct or not) subproperty of property *p*.
- C_EXT(c, x) iff resource *x* is in the proper extent – is a direct instance – of class *c*.
- P_EXT(x, p, y) iff the ordered pair (*x*, *y*) (formed from resources *x*, *y*) is in the proper extent – is a direct instance – of property *p*.

The translation of a specific RDF/S schema in SWLF is straightforward: the predicates CLASS, PROP, C_SUB and P_SUB get instantiated for every class, property and direct subclass/subproperty relationship. For example, some of the facts we generate for the RDF/S schema of **Fig. 1** are:

```
CLASS(Artist), PROP(Artist,creates,Artifact), C_SUB(Painter,Artist),
P_SUB(paints,creates), P_SUB(sculpts,creates)
```

Definition 2. Disjunctive embedded dependencies (DEDs) are FOL formulas of the following general form:

$$\forall \overline{x}\left[\varphi(\overline{x}) \rightarrow \bigvee_{i=1}^{l} \exists \overline{y_i} \varphi_i^{'}(\overline{x}, \overline{y_i})\right]$$

where $\overline{x}$, $\overline{y_i}$ are tuples of variables and φ, $\varphi_i^{'}$ are conjunctions of relational atoms of the form $R(\omega_1, ..., \omega_l)$ and equality atoms of the form $\omega=\omega^{'}$, where $\omega_1, ..., \omega_l, \omega, \omega^{'}$ are variables or constants; φ may be the empty conjunction.

In particular, in order to capture the intended meaning of an RDF/S schema two types of constraints are considered in SWLF; for each predicate, many constraints existentially quantifying its variables and one constraint universally quantifying them. The constraints of the former type provide partial (incomplete) knowledge of the RDF/S schema and have the general form:

$$\exists \overline{y} \varphi^{'}(\overline{y}) \qquad (1)$$

where $\overline{y}$ is a tuple of variables and $\varphi^{'}$ is a conjunction of one SWLF predicate and equalities between the variables of one of the SWLF predicates and constants. This form is equivalent to $\varphi'(\overline{y})$, where φ' is an SWLF predicate and $\overline{y}$ is a tuple of constants, and corresponds to the form of the facts generated for an RDF/S schema. On the other hand, the constraints of the latter type state which are all the classes, the properties and the subsumption relationships, and, thus, provide along with those of the previous type a complete knowledge of the RDF/S schema. Their general form is:

$$\forall\overline{x}\left[\varphi(\overline{x}) \rightarrow \bigvee_{i=1}^{l} \varphi_i'(\overline{x})\right] \quad (2)$$

where $\overline{x}$ is a tuple of variables, φ is an SWLF predicate and each φ_i' is a conjunction of equalities between the predicate's variables and constants. Constraints of this form capture also the reflexivity and transitivity of class and property subsumption relationships. **Table 2** shows the constraints of form (2) introduced for the P_SUB predicate given the RDF/S schema of **Fig. 1**.

Table 2. Example of constraints introduced for P_SUB given the RDF/S schema of **Fig. 1**

Constraint universally quantifying P_SUB's variables
∀subP ∀prop (P_SUB(subP, prop) → (subP="creates" ∧ prop="creates") ∨ (subP="paints" ∧ prop="creates") ∨ (subP="paints" ∧ prop="paints") ∨ (subP="sculpts" ∧ prop="creates") ∨ (subP="sculpts" ∧ prop="sculpts"))

In order to capture the semantics of the RDF/S data model, i.e., of every RDF/S schema and description base, we consider a set of constraints, called δ_{Mod}.

Definition 3. δ_{Mod} is the set of the following disjunction-free DEDs:

- Every resource in the extent of a class implies the existence of the corresponding class: $\forall c,x\ C\_EXT(c,x) \rightarrow CLASS(c)$
- The subclass relationship relates classes: $\forall c,d\ C\_SUB(c,d) \rightarrow CLASS(c) \wedge CLASS(d)$
- The domain and range of every property is a class: $\forall a,p,b\ PROP(a,p,b) \rightarrow CLASS(a) \wedge CLASS(b)$
- The domain and range of every property is unique: $\forall a,p,b,c,q\ d\ PROP(a,p,b) \wedge PROP(c,q,d) \wedge p=q \rightarrow a=c \wedge b=d$
- Every statement in the extent of a property implies the existence of the corresponding property: $\forall x,p,y\ P\_EXT(x,p,y) \rightarrow \exists c,d\ PROP(c,p,d)$
- The subproperty relationship relates properties: $\forall p,q\ P\_SUB(p,q) \rightarrow \exists a,b,c,d\ PROP(a,p,b) \wedge PROP(c,q,d)$
- Every class is a subclass of itself: $\forall c\ CLASS(c) \rightarrow C\_SUB(c,c)$
- The subclass relationship is transitive: $\forall a,c,e\ C\_SUB(e,c) \wedge C\_SUB(c,a) \rightarrow C\_SUB(e,a)$
- Every property is a subproperty of itself: $\forall c,p,d\ PROP(c,p,d) \rightarrow P\_SUB(p,p)$
- The subproperty relationship is transitive: $\forall p,q,r\ P\_SUB(p,q) \wedge P\_SUB(q,r) \rightarrow P\_SUB(p,r)$
- A class is both subclass and superclass of itself: $\forall a,c\ C\_SUB(a,c) \wedge C\_SUB(c,a) \rightarrow a=c$

- A property is both subproperty and superproperty of itself : $\forall p,q\ P\_SUB(p,q) \wedge P\_SUB(q,p) \rightarrow p=q$
- In a valid RDF description schema the domain (range) of every subproperty is subsumed by the domain (range) of its superproperty: $\forall a,p,b,c,q,d\ P\_SUB(q,p) \wedge PROP(a,p,b) \wedge PROP(c,q,d) \rightarrow C\_SUB(c,a) \wedge C\_SUB(d,b)$
- In a valid RDF description base the subject/object resources in every statement are (direct or indirect) instances of the property's domain/range classes: $\forall a,p,b,x,y\ PROP(a,p,b) \wedge P\_EXT(x,p,y) \rightarrow \exists c,d\ C\_SUB(c,a) \wedge C\_SUB(d,b) \wedge C\_EXT(c,x) \wedge C\_EXT(d,y)$

It should be stressed that, compared to the RDF/S Semantics given in [14], SWLF *(i)* distinguishes between the different RDF/S abstraction layers (data, schema and metaschema), *(ii)* enforces that a property's domain and range are always defined and unique, *(iii)* does not allow the existence of cycles in the class and property hierarchies, *(iv)* states that the set inclusion of the domain and range are preserved for specialized properties and *(v)* demands that in each statement the subject and object resources are (direct or not) instances of the domain and range classes of the property, respectively. These additional constraints are employed to reason over queries (and not on data as in [14]), clarify the semantics of classes and properties and decrease the complexity of RDF/S query containment and minimization.

In this context, we consider the following two additional sets of constraints capturing (partially or completely) the semantics of a particular RDF/S schema:

Definition 4. δ_{RDF} is the set of disjunction-free DEDs consisting of δ_{Mod} and the constraints of form (1).

Definition 5. Δ_{RDF} is the set of DEDs consisting of δ_{Mod} and the constraints of forms (1) and (2).

Having in mind the FOL predicates *R* and the aforementioned sets of constraints, the formal definitions of an RDF/S *description base* and a *description schema* are:

Definition 6. An RDF/S (description) schema *DS* in SWLF is an instantiation of the relational schema $R_S=\{CLASS, PROP, C\_SUB, P\_SUB\}$ satisfying δ_{Mod}.

Definition 7. An RDF/S description base *DB* in SWLF given a *DS* is an instantiation of the relational schema $R_B=\{C\_EXT, P\_EXT\}$ satisfying δ_{Mod}.

3 RDF/S Query Languages' Fragments

All RDF/S query languages provide pattern matching facilities against schema and/or data graphs. The main difference in their expressiveness is related to their ability to support either exact or extended pattern matching by taking into account the subsumption relationship of classes and properties defined in an RDF/S schema. When only exact matching is supported, patterns of schema-agnostic RDF/S query languages (e.g., SPARQL [21]) can be divided into two main fragments: The first includes patterns for pure data matching given as input the schema classes and properties (i.e., as in relational queries), while the latter includes patterns that

Table 3. RDF/S query patterns categorization

		Property Patterns	**Class Patterns**
$\mathrm{RQL}_{\mathrm{UCQ}}$		{X; \$C}@P{Y; \$D}	\$C{X; \$D}
		{\$C}@P{\$D}	\$C{\$D}
		{X}@P{Y}	\$C{X}
		{X}^p{Y}	^c{X}
	$\mathrm{RQL}_{\mathrm{CORE}}$	{X}p{Y}	c{X}
		{X; c}p{Y; d}	c{X; d}

arbitrary mix schema and data querying. When both exact and extended matching is supported, patterns of schema-aware RDF/S query languages (e.g., RQL [15]) can be similarly divided into the previous two fragments while the latter also includes patterns for exact matching of schema and data graphs.

Table 3 presents the basic RDF/S class and property patterns of RQL (capital letters denote variables, and small letters denote constants), as well as introduces the fragments that these patterns belong to. With the exception of the RQL distinction between exact (denoted with '^') and extended pattern matching for class (^c{X} and c{X}) and property ({X}^p{Y} and {X}p{Y}) instances, all the other patterns are encountered in the majority of the RDF/S query languages. In this context, the SQO algorithms presented in this paper for the two most expressive previous RQL fragments [12] can be naturally applied to other RDF/S query languages as long as the appropriate translations of their patterns to SWLF are provided.

In particular, in this paper we focus on unions of RQL conjunctive queries, called *RQL_{UCQ},* that are defined analogously to unions of relational conjunctive queries; the only difference lies to the fact that RQL class/property patterns are used instead of simple relational predicates. Indeed, according to the declarative semantics given in [15], RQL patterns have the same meaning as conjunctions of relational atoms.

Definition 8. An RQL conjunctive query is a FOL formula of the following form:

$$ans(\overline{u}) :- \ldots, E_i(\overline{u_i}), \ldots, u_m = u_n$$

where $\overline{u}$ is a tuple of variables or constants, $E_i(\overline{u_i})$'s are class/property patterns (see **Table 3**) and $u_m = u_n$'s are equalities between variables and/or constants. Each $\overline{u_i}$ involves the variables X_i, $\$C_i$, $@P_i$, Y_i, $\$D_i$ – where $@P_i$ is a property variable, $\$C_i$ and $\$D_i$ are class variables, X_i and Y_i are resource variables – or a subset of them.

Note that RQL conjunctive queries must be *safe*, i.e., their variables must be range restricted as for relational queries [2]. By extending the above formalism, we get the following definition.

Definition 9. $\mathrm{RQL}_{\mathrm{UCQ}}$ queries have the form: $\bigcup_k Q_k$

where Q_k's are RQL conjunctive queries whose heads have the same type.

Definition 10. *RQL_{CORE}* is a subset of $\mathrm{RQL}_{\mathrm{UCQ}}$ including patterns for pure data matching (see **Table 3**).

The complete list of the property patterns for both RQL_{UCQ} and RQL_{CORE} fragments is given in the Appendix (class patterns are defined in a similar way). As we will see in the sequel, the gain from limiting the expressiveness of RQL_{UCQ} queries to RQL_{CORE} is double. First of all, the partial knowledge of the RDF/S schema offered from δ_{RDF} suffices to solve the containment and minimization problems for the RQL_{CORE} (i.e., we do not need complete schema information). Additionally, considering only the information provided from δ_{RDF} leads to lower execution costs of the containment, equivalence and minimization algorithms, which stems from the fact that the involved *chase* algorithm considers only disjunction-free constraints (see Section 4).

RQL_{UCQ} (RQL_{CORE}) queries can be translated into unions of relational conjunctive queries expressed in terms of SWLF (see Appendix for the complete list of property pattern translation). The translation is straightforward: given an RQL_{UCQ} (RQL_{CORE}) query, class/property patterns get substituted by the corresponding SWLF predicates.

Example 1. Take a look at the translation of the RQL_{UCQ} query[1] retrieving cubists who have painted artefacts that are exhibited.

```
SELECT     X
FROM       {X; Cubist}paints{Y}, {Y}exhibited
```

By replacing the constants found in the patterns with variables and adding the corresponding equalities we get:

```
SELECT     X
FROM       {X; $C}@P1{Y}, {Y}@P2
WHERE      $C=Cubist and @P1=paints and @P2=exhibited
```

Now, the query can be rewritten in SWLF as follows:

```
ans(X):- {X; $C}@P1{Y}, {Y}@P2, $C=Cubist, @P1=paints,
@P2=exhibited
```

The corresponding SWLF query is produced by employing P_SUB to navigate through the subproperties of *paints* and *exhibited*, P_EXT to retrieve the direct instances of these subproperties, C_SUB to retrieve *Cubist*'s subclasses and C_EXT to retrieve the direct instances of these subclasses. Finally, C_SUB and PROP are used to ensure that *Cubist* is a subclass of *paint*'s domain.

```
ans(x):- PROP(a,p1,b), P_SUB(q1,p1), P_EXT(x,q1,y), C_SUB(c,a),
C_SUB(e,c), C_EXT(e,x), P_SUB(q2,p2), P_EXT(y,q2,z),
c="Cubist", p1="paints", p2="exhibited"
```

4 RQL Query Containment and Equivalence

For the RQL_{UCQ} and RQL_{CORE} fragments we define containment and equivalence as follows:

Definition 11. An RQL_{UCQ} (RQL_{CORE}) query Q_1 is contained in an RQL_{UCQ} (RQL_{CORE}) query Q_2 ($Q_1 \sqsubseteq Q_2$) given an RDF description schema *DS* iff for every RDF

[1] For simplicity, in all RQL queries presented in this paper namespaces are disregarded.

description base *DB* conforming to *DS*, the result of Q_1 is contained in that of Q_2 ($\forall$DB $Q_1(DB) \subseteq Q_2(DB)$).

Definition 12. An RQL_{UCQ} (RQL_{CORE}) query Q_1 is equivalent to an RQL_{UCQ} (RQL_{CORE}) query Q_2 ($Q_1 \equiv Q_2$) given an RDF description schema *DS* iff for every RDF description base *DB* conforming to *DS*, the result of Q_1 is equivalent to that of Q_2 ($\forall$DB $Q_1(DB) \equiv Q_2(DB)$).

As stated previously, we reduce the RQL query containment and equivalence problems to the relational ones under constraints. In this section we introduce the *chase* algorithm and present how it can be employed in our RQL_{UCQ} (RQL_{CORE}) containment and equivalence checking.

4.1 Chase Algorithm

The core chase consists of a sequential execution of a number of chase steps. For example, given the constraint $\forall x \forall y\ A(x, y) \rightarrow B(x)$ and the query Q(x) :- A(x, y), the chase step leads to query Q(x) :- A(x, y), B(x). When no more chase steps can apply the chase ends and the query outputted is called the *universal plan*.

Unfortunately, the chase with an arbitrary set of DEDs is not guaranteed to terminate. However, in [9] the authors introduced a syntactic restriction, namely *stratified-witness*, which ensures termination of the chase under a set of disjunction-free DEDs. When a set of DEDs respects stratified-witness, no sequence of chase steps can force the chase to diverge. Stratified-witness can be extended in order to handle constraints that use disjunction, too (see [22] for further details). The key idea lies on the splitting of disjunctive constraints into disjunction-free ones and checking whether one of the possible combinations of disjunction-free constraints leads to an endless execution of chase steps by checking them for stratified-witness.

In [7] the author proves soundness of the chase-based containment algorithm for conjunctive queries under a set of DEDs. In the sequel, we extend this algorithm to unions of conjunctive queries:

Theorem 1. Given two unions of conjunctive queries Q_1, Q_2 and a set *D* of DEDs, assume that the chase of Q_1 with *D* terminates rendering the universal plan U_1. Then, Q_1 is contained in Q_2 under *D* ($Q_1 \subseteq_D Q_2$) iff[2] for every *i* there is a *j* such that U_{1i} is contained in Q_{2j}, i.e., there is a containment mapping from Q_{2j} into U_{1i}.

Moreover, it is obvious that two queries Q_1, Q_2 are equivalent under a set *D* of constraints iff $Q_1 \subseteq_D Q_2$ and $Q_2 \subseteq_D Q_1$. Therefore, at least one and at most two containment checks are needed in order to decide the equivalence of queries under constraints.

4.2 Checking Containment and Equivalence of RQL_{CORE} Queries

The algorithm for checking whether an RQL_{CORE} query is contained in another is based on Theorem 1. It takes as input the two SWLF queries and δ_{RDF}, which is a set

[2] In contrast to [7] and without loss of generality, after each chase step we check the query for inconsistencies, i.e. equalities between distinct constants.

of DEDs that behaves as if stratified-witness[3] was present and, therefore, ensures the termination of chase and, thus, soundness of the containment check.

Example 2. Assume that we want to check the containment of the query retrieving people having painted

```
SELECT     X
FROM       {X}paints
```

in the query returning all artists

```
SELECT     X
FROM       Artist{X}
```

The queries are translated, respectively, into SWLF as follows:

ans(x):-P_SUB(q,p), P_EXT(x,q,y), p="paints"

ans(x):-C_SUB(c,a), C_EXT(c,x), a="Artist"

By chasing the first query with the basic constraint $\forall p \forall q\ P\_SUB(q, p) \rightarrow \exists a_1 \exists a_2 \exists b_1 \exists b_2\ PROP(a_2, q, b_2) \wedge PROP(a_1, p, b_1)$ we get the query[4]:

ans(x):-$\underline{\text{P\_SUB(q,p)}}$, P_EXT(x,q,y), **PROP($a_1$,p,$b_1$)**, **PROP($a_2$,q,$b_2$)**, p="paints"

The next chase step involves the first domain/range constraint:

ans(x):-$\underline{\text{P\_SUB(q,p)}}$, P_EXT(x,q,y), $\underline{\text{PROP(}a_1\text{,p,}b_1\text{)}}$, $\underline{\text{PROP(}a_2\text{,q,}b_2\text{)}}$, **C_SUB($a_2$,$a_1$)**, **C_SUB($b_2$,$b_1$)**, p="paints"

The following one involves the second domain/range constraint:

ans(x):-P_SUB(q,p), $\underline{\text{P\_EXT(x,q,y)}}$, PROP($a_1$,p,$b_1$), $\underline{\text{PROP(}a_2\text{,q,}b_2\text{)}}$, C_SUB($a_2$,$a_1$), C_SUB($b_2$,$b_1$), **C_SUB(c,$a_2$)**, **C_SUB(d,$b_2$)**, **C_EXT(c,x)**, **C_EXT(d,y)**, p="paints"

After a number of chase steps we reach the following (incomplete) universal plan:

ans(x):-P_SUB(q,p), P_EXT(x,q,y), PROP(a_1,p,b_1), PROP(a_2,q,b_2), C_SUB(a_2,a_1), C_SUB(b_2,b_1), C_SUB(c,a_2), C_SUB(d,b_2), C_EXT(c,x), C_EXT(d,y), C_SUB(c,a_1), C_SUB(h,g), C_SUB(c,g), p="paints", a_1=h="Painter", b_1="Painting", g="Artist"

Since there is a containment mapping from the second input query to the chased query ($\{c \rightarrow c, a \rightarrow g, x \rightarrow x\}$), the first query is contained in the second one.

Using the same algorithm we can prove that the query of Example 1 is contained in the first query of Example 2. Having reduced the RQL_{CORE} containment problem to a relational one, the equivalence problem gets reduced to the relational one as well.

4.3 Checking Containment of RQL_{UCQ} Queries

The same algorithm can be used for checking containment between RQL_{UCQ} queries. It takes as input the two SWLF queries and the set of constraints Δ_{RDF}, which ensures the termination of chase and, thus, soundness of the containment check.

[3] δ_{RDF} (and Δ_{RDF}) is a set of DEDs not satisfying stratified-witness. However, it does not allow the introduction of an infinite number of fresh variables [22].

[4] The predicates triggering the chase step are underlined while the introduced ones are given in bold.

Example 3. Assume that we want to check the containment of the following query retrieving people who have painted artefacts that are exhibited somewhere

```
SELECT    X
FROM      {X}paints{Y}, {Y}exhibited
```

in the query returning people having exclusively painted (i.e., in the proper interpretation of *paints*) something

```
SELECT    X
FROM      {X}^paints
```

The input queries are translated respectively into SWLF as follows:

$\texttt{ans(x):-P\_SUB(}q_1\texttt{,}p_1\texttt{), P\_EXT(x,}q_1\texttt{,y), P\_SUB(}q_2\texttt{,}p_2\texttt{),}$
$\texttt{P\_EXT(y,}q_2\texttt{,z), }p_1\texttt{="paints", }p_2\texttt{="exhibited"}$

and

```
ans(x):-P_EXT(x,p,y), p="paints"
```

After a number of chase steps the first query reaches the (incomplete) universal plan[5]:

$\texttt{ans(x):- P\_SUB(}q_1\texttt{,}p_1\texttt{), P\_EXT(x,}q_1\texttt{,y), P\_SUB(}q_2\texttt{,}p_2\texttt{),}$
$\texttt{P\_EXT(y,}q_2\texttt{,z), }p_1\texttt{=}q_1\texttt{="paints", }p_2\texttt{=}q_2\texttt{="exhibited"}$

There is a containment mapping from the second input query to the chased query ($\{p \rightarrow q_1, x \rightarrow x, y \rightarrow y\}$). Therefore, the first query is contained in the second one.

As with RQL_{CORE}, the RQL_{UCQ} equivalence problem is also reduced to the relational one.

In the beginning of this section we have claimed that containment of RQL_{UCQ} queries can be checked in presence of Δ_{RDF}. It should be stressed that any restriction of Δ_{RDF}, either by employing δ_{RDF} or by considering the set of constraints that excludes from Δ_{RDF} the constraints of form (1), affects the soundness of the containment (see [22] for further details). The same stands for the minimization algorithm presented in the next section.

The complexities of both containment (equivalence) algorithms depend on the cost of the chase algorithm to reach the universal plan and the cost of the simple containment check at the end. In this context, the chase depends on the set of constraints considered (δ_{RDF} for RQL_{CORE} and Δ_{RDF} for RQL_{UCQ}) and on the size of the input queries (in presence or not of union). Note that the chase with disjunction-free constraints satisfying stratified-witness (or behaving as if satisfying it, like δ_{RDF}) is NP-complete [20].

5 RQL Query Minimization

In this section, we detail how we can minimize RQL_{UCQ} (RQL_{CORE}) queries using the *backchase* algorithm. Furthermore, we highlight how the produced minimal SWLF queries can be translated both to schema-aware RDF/S query languages, like RQL, and to schema-agnostic languages, like SPARQL.

Definition 13. Given an RDF description schema *DS* an RQL_{UCQ} (RQL_{CORE}) query *Q* gets minimized when replaced with a minimal equivalent query *SQ* ($\forall$DB Q(DB)$\equiv$SQ(DB)).

[5] If we applied all possible chase steps, the constraints would introduce union in the universal plan.

A *minimal* RQL_{UCQ} (RQL_{CORE}) query uses fewer and/or simpler RQL patterns than the original query. The intuition is that a class pattern is simpler than a property one; a pattern involving proper interpretations (for RQL_{UCQ}) and/or fewer variables is simpler than one involving extended interpretations and/or more variables. The above hypotheses are made by taking into account that the evaluation of a simpler pattern is more efficient that the original one.

5.1 Backchase Algorithm

The core algorithm is the backchase, which, given a query's universal plan, checks all its subqueries for minimality and equivalence to the original query using chase. According to the following theorem introduced in [9] the backchase is guaranteed to find all minimal equivalent subqueries when the chase terminates and, thus, ensures completeness of the minimization algorithm.

Theorem 2 [9]. Given a union of conjunctive queries *Q* and a set *C* of DEDs, if the chase of *Q* with *C* terminates yielding the universal plan of *U*, all *C*-minimal reformulations of Q are subqueries of *U*.

5.2 Minimization of RQL_{CORE} Queries Using Schema Knowledge

In order to minimize an RQL_{CORE} query the universal plan of the original SWLF query and δ_{RDF} are given as input. Since the chase with δ_{RDF} terminates, the backchase always finds all minimal equivalents of an RQL_{CORE} query.

Example 4. Assume the following query

```
SELECT     X
FROM       Cubist{X}, Painter{X}
```

which was introduced in Example 3. Its SWLF translation will chase to:

```
ans(x):-C_SUB(c1,a1), C_EXT(c1,x), C_SUB(c2,a2), C_EXT(c2,x),
C_SUB(e,d), a1=d="Painter", a2=e="Cubist"
```

If we examine its subquery retrieving the extended interpretation of *Cubist*

```
ans(x):-C_SUB(c,a), C_EXT(c,x), a="Cubist"
```

we will conclude that it is δ_{RDF}-minimal and δ_{RDF}-equivalent to the query given as input. Thus, this query will be produced by our minimization algorithm.

It is worth noticing that RQL_{CORE} queries demonstrate a very interesting and useful feature: they have only one minimal equivalent! As we will explain in the next subsection, more than one minimal query can occur only by replacing extended interpretations with proper ones, which are not supported by RQL_{CORE}.

5.3 Minimization of RQL_{UCQ} Queries Using Schema Knowledge

Similarly, minimization of RQL_{UCQ} is always successful since the set of constraints employed in this case – Δ_{RDF} – guarantees termination.

Example 5. Assume the query of Example 4. If considered as an RQL_{UCQ} query, it will minimize to the query retrieving the proper interpretation of *Cubist*

```
ans(x):-C_EXT(c,x), c="Cubist"
```

The difference in the minimal query is due to the fact that the backchase has the additional knowledge that *Cubist* has no subclass than itself. As we will see in the sequel, unlike RQL_{CORE}, RQL_{UCQ} queries may have more than one minimal equivalent.

Example 6. Assume the query

```
SELECT     $A, X
FROM       $A{X; Artist}
```

and its SWLF translation

```
ans(a,x):-C_SUB(c,a), C_SUB(e,c), C_EXT(e,x), c="Artist"
```

If we execute the C&B algorithms, we will find three (!) minimal equivalent queries:

```
(1st) ans(a,x):-C_SUB(e,a), C_EXT(e,x), a="Artist"
(2nd) ans(a,x):-C_EXT(a,x), a="Artist"
  ∪ ans(a,x):-C_EXT(e,x), a="Artist", e="Sculptor"
  ∪ ans(a,x):-C_SUB(e,c), C_EXT(e,x), a="Artist", c="Painter"
(3rd) ans(a, x):- C_EXT(a, x), a="Artist"
  ∪ ans(a, x):-C_EXT(e, x), a="Artist", e="Sculptor"
  ∪ ans(a, x):-C_EXT(e, x), a="Artist", e="Painter"
  ∪ ans(a, x):-C_EXT(e, x), a="Artist", e="Cubist"
```

The most interesting minimal queries are the first and third ones. In the first one redundancy has been removed without resolving the navigational part occurring from traversing the subclass hierarchy of *Artist*; this is why the extended interpretation of *Artist* is used. On the contrary, in the third minimal query schema information has been completely unfolded, introducing a union involving only the proper interpretations of *Artist*'s subclasses. When the former will be executed against an RDF/S store, it will still require schema navigation, while the latter contains all necessary schema information to retrieve the resources and, thus, it can be used by schema-agnostic languages. The second query lies somewhere in the middle; since both proper and extended interpretations are used, a part of the schema information has been unfolded, while some other has not. This form seems useful when the results of some of the constituent conjunctive queries are already cached.

In general, the number of minimal queries depends on the constraints considered, i.e., the size of the RDF/S schema, and the query given as input. As they grow, the number of minimal equivalents considerably increases. Every RQL_{UCQ} query has one minimal equivalent query where schema information is completely unfolded. This means that if the original query does not involve pattern matching at the data level, minimization practically answers the original query; the result is a constant query, i.e., a query were only equalities appear in the body. Apart from it, there usually exists one minimal query where the unfolding has not introduced union and several ones where partial unfolding has taken place.

5.4 Minimization of RQL_{UCQ} Queries by Ignoring Schema Knowledge

Our minimization technique can, also, be used for minimizing RQL patterns in their general form without taking into consideration specific RDF/S schemas [22]; therefore the chase in this case considers only δ_{Mod}.

Example 7. Assume the RQL_{UCQ} query

```
ans(X, @P, Y):-{X; $C}@P{Y; $D}, cond(X, @P, Y)
```

involving the pattern we want to simplify and a dummy predicate *cond* stating the conditioned variables. The equivalent SWLF query is:

```
ans(x,p,y):-PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y),
C_SUB(c,a), C_SUB(d,b), C_EXT(c,x), C_EXT(d,y), cond(x,p,y)
```

If we inspect the universal plan of the query above during backchase, we will reach the subquery

```
ans(x,p,y):- P_SUB(q,p), P_EXT(x,q,y), cond(x,p,y)
```

which is both δ_{Mod}-minimal and δ_{Mod}-equivalent. Interestingly, it corresponds to the RQL_{UCQ} query

```
ans(X, @P, Y):-{X}@P{Y}, cond(X, @P, Y)
```

Thus, the pattern *{X; $C}@P{Y; $D}* gets simplified to pattern *{X}@P{Y}* when only variables *X*, *@P*, *Y* are either conditioned or projected.

5.5 Backward Translation

As a matter of fact, in some cases we would like to restore the initial form (i.e., RQL_{UCQ} or RQL_{CORE}) of a minimal query expressed in terms of SWLF. In the case of RQL_{CORE} the backward translation is simple since only two RQL patterns, in particular *$C{X}* and *{X}@P{Y}*, may appear in the body of an RQL_{CORE} minimal query. On the contrary, the translation becomes somehow more complicated for RQL_{UCQ}. Initially, we need to identify simple patterns in the SWLF query and combine them in order to form more complex ones. For example, the patterns *{X}@P*, *{$C}@P*, *$C{X}* correspond to the RQL pattern *{X; $C}@P*. For both RQL_{UCQ} and RQL_{CORE} minimal queries we have to reduce the number of employed variables and replace with constants as many variables as possible by using the equalities between the variables and constants. The following example illustrates the backward translation of an SWLF query into RQL_{UCQ}.

Example 8. In the first phase, the first minimal query of Example 6 translates into:

```
SELECT     $A, X
FROM       $A{X}
WHERE      $A=Artist
```

The second phase does not affect the query. Finally, by incorporating the only available equality in the FROM and SELECT clauses we get the RQL_{UCQ} query

```
SELECT     Artist, X
FROM       Artist{X}
```

There are two tricky issues regarding the backward translation. Firstly, δ_{Mod} implies the equivalence of the predicates CLASS(c), C_SUB(c, c) and PROP(a, p, b), P_SUB(p, p). However, C_SUB(c, c) and P_SUB(p, p) result in redundant processing from an RDF/S query engine. Additionally, there is no RQL pattern corresponding to P_SUB(p, p). Thus, for these predicates we employ the translations of CLASS(c) – i.e., *$C* – and PROP(a, p, b) – i.e., *@P* -, respectively. Secondly, although they do not belong to the RQL_{UCQ} fragment, the functions *domain(@P)* and *range(@P)* should be used for some SWLF queries due to the lack of an RQL pattern that would explicitly impose a restriction on a property's domain/range.

Example 9. Assume the RQL_{UCQ} query of Example 6. The translation of the third minimal query into RQL and SPARQL[6] is given below.

SPARQL	RQL
`SELECT ?C ?X`	`SELECT Artist, X`
`WHERE {{?X rdf:type :Artist . ?C rdf:type rdfs:Class .`	`FROM   ^Artist{X}`
`  FILTER ?C = :Artist}`	`UNION`
`UNION`	`SELECT Artist, X`
`WHERE {{?X rdf:type :Sculptor . ?C rdf:type rdfs:Class .`	`FROM  ^Sculptor{X}`
`  FILTER ?C = :Artist}`	`UNION`
`UNION`	`SELECT Artist, X`
`WHERE {{?X rdf:type :Painter . ?C rdf:type rdfs:Class .`	`FROM   ^Painter{X}`
`  FILTER ?C = :Artist}`	`UNION`
`UNION`	`SELECT Artist, X`
`WHERE {{?X rdf:type :Cubist . ?C rdf:type rdfs:Class .`	`FROM   ^Cubist{X}`
`  FILTER ?C = :Artist}`	

The complexities of all the previous minimization algorithms depend on the backchase which, in turn, depends on the chase and the simple containment check – employed in order to reach the universal plan and check all its subqueries for minimality. The full optimization corresponds to an exponential number of NP-complete problems [20].

6 Summary and Future Work

In this paper we studied SQO of patterns supported by expressive RDF/S query languages, like RQL. Nevertheless, our results are valid for less expressive query languages, too. In order to deal with the SQO problem, we advocate a logic framework that enables to reduce the containment and minimization problems for unions of RDF/S conjunctive queries into relational equivalents.

In particular, the C&B algorithms, which we employed for RDF/S SQO, were initially developed in the context of conjunctive queries issued against relational schemas and matched (exclusively) against data by taking into account embedded dependencies [2] (for capturing key and foreign key constraints, as well as views). In our context, we consider unions of conjunctive queries for checking containment and minimization of queries built on expressive RDF/S query patterns asking for both schema and data matching over class (or property) subsumption hierarchies, and constraints under the form of disjunctive embedded dependencies. In contrast to relational queries, RDF/S queries usually contain a schema navigational part (e.g., in order to obtain the extended instances of a class, we need to consider the instances of all its direct and transitive subclasses). Therefore, the goal of RDF/S SQO is twofold: (a) the schema navigational part must be pruned as much as possible and (b) redundant data access should be eliminated as in the case of traditional SQO in relational databases. In conjunction with the aforementioned variation of SQO for the Semantic Web (SW) is the fact that the RQL_{UCQ} minimization algorithm always generates a minimal query where no further schema querying is needed in order to answer it.

[6] For simplicity reasons namespaces' definitions are disregarded.

As a future work we are planning to study SQO of ontology constructs originating from more expressive SW languages, such as OWL's inverse properties as well as disjointness of class and property interpretations. Moreover, we plan to study SQO of richer RDF/S query fragments involving functions – like domain, range, subclassof, subpropertyof and aggregate ones – as well as nested queries.

Acknowledgements

We would like to thank Alin Deutsch and Nicola Onose for fruitful discussions on relational and XML SQO.

References

[1] Nikos Athanasis, Vassilis Christophides, and Dimitris Kotzinos. *Generating on the Fly Queries for the Semantic Web: The ICS-FORTH Graphical RQL Interface (GRQL).* In Proceedings of the 3rd International Semantic Web Conference, Japan, 2004.

[2] Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of Databases.* Addison-Wesley, 1995.

[3] Tim Bray, Eve Maler, Jean Paoli, and C. M. Sperberg-McQueen. *Extensible Markup Language (XML) 1.0.* W3C Recommendation, 6 October 2000.

[4] Francois Bry. *Query Answering in Information Systems with Integrity Constraints.* In Proceedings of the 1st Working Conference on Integrity and Internal Control in Information Systems: Increasing the confidence in Information Systems, Zurich, 1997.

[5] Tim Berners-Lee, James Hendler, and Ora Lassila. *The Semantic Web: A new form of Web content that is meaningful to computers will unleash a revolution of new possibilities.* Scientific American, 17 May 2001. Available at http://www.scientificamerican.com/print_version.cfm?articleID=00048144-10D2-1C70-84A9809EC588EF21

[6] Upen S. Chakravarthy, John Grant, Jack Minker. *Logic-Based Approach to Semantic Query Optimization.* ACM Transactions on Database Systems 15(2): 162-207 (1990)

[7] Alin Deutsch. *XML Query Reformulation over Mixed and Redundant Storage.* PhD Thesis, University of Pennsylvania, 2002.

[8] Alin Deutsch, Lucian Popa, and Val Tannen. *Physical Data Independence, Constraints and Optimization with Universal Plans.* In Proceedings of the 25th International Conference on Very Large Databases (VLDB), Edinburgh, 1999.

[9] Alin Deutcsh and Val Tannen. *Reformulation of XML Queries and Constraints.* In Proceedings of the 9th International Conference on Database Theory (ICDT), Italy, 2003.

[10] Xin Dong, Alon Y. Halevy, and Igor Tatarinov. *Containment of Nested XML Queries.* In Proceedings of 30th International Conference on Very Large Databases (VLDB), Toronto, Canada, 2004.

[11] John Grant, Jarek Gryz, Jack Minker, and Louiqa Raschid. *Semantic Query Optimization for Object-Databases.* In Proceedings of the 13th International Conference on Data Engineering, Birmingham U.K, 1997.

[12] Peter Haase, Jeen Broekstra, Andreas Eberhart, and Raphael Volz. *A Comparison of RDF Query Languages.* In Proceedings of the 3rd International Semantic Web Conference, Japan, 2004.

[13] Frank Van Harmelen and Deborah L. McGuinness. *OWL Web Ontology Language Overview.* W3C Recommendation, 10 February 2004.

[14] Patrick Hayes. *RDF Semantics.* W3C Recommendation, 10 February 2004.

[15] Gregory Karvounarakis, Aimilia Magkanaraki, Sofia Alexaki, Vassilis Christophides, Dimitris Plexousakis, Michel Scholl, and Karsten Tolle. *Querying the Semantic Web with RQL*. Computer Networks 42(5): 617-640, 2003.

[16] Alon Levy and Yehoshua Sagiv. Semantic Query Optimization in Datalog Programs. In Proceedings of the 8th International Conference on Data Engineering, Tempe, Arizona 1992.

[17] Frank Manola and Eric Miller. *RDF Primer*. W3C Recommendation, 10 February 2004.

[18] Gerome Miklau and Dan Suciu. *Containment and equivalence for an XPath fragment*. In Proceedings of the 21st ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. Madison, Wisconsin, 2002.

[19] Hwee Hwa Pang, HongJun Lu, and Beng Chin Ooi. An Efficient Semantic Query Optimization Algorithm. In Proceedings of the 7th International Conference on Data Engineering, Japan, 1991.

[20] Nicola Onose. *Extensions of the Relational Chase*. Project Report of End of Studies, 2005.

[21] Eric Prud'hommeaux, and Andy Seaborne. *SPARQL Query Language for RDF*. W3C Working Draft, 19 April 2005.

[22] Giorgos Serfiotis. Optimizing and Reformulating RQL Queries on the Semantic Web. Master's Thesis, University of Crete, 2005.

[23] Heiner Stuckenschmidt. *Similarity-Based Query Caching*. In Proceedings of the 6th International Conference on Flexible Query Answering Systems, Lyon, 2004.

[24] Cong Yu and Lucian Popa. *Constraint-Based XML Query Rewriting For Data Integration.* In Proceedings of the ACM SIGMOD International Conference on Management of Data, Paris, 2004.

Appendix: Translations of RQL$_{UCQ}$(RQL$_{CORE}$) Patterns into SWLF

Property Pattern	Translation	Fragment
@P ^@P	PROP(a,p,b)	RQLCORE when $C=c, $D=d, @P=p RQLUCQ
{X; $C}@P{Y; $D} {$C}@P{Y; $D} {X; $C}@P{$D}	PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y), C_SUB(c,a), C_SUB(d,b), C_SUB(e,c), C_SUB(f,d), C_EXT(e,x), C_EXT(f,y)	
{X; $C}@P{Y} {$C}@P{Y} {X; $C}@P	PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y), C_SUB(c,a), C_SUB(e,c), C_EXT(e,x)	
{X}@P{Y; $D} {X}@P{$D} @P{Y; $D}	PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y), C_SUB(d,b), C_SUB(f,d), C_EXT(f,y)	
{X}@P{Y} {X}@P @P{Y}	P_SUB(q,p), P_EXT(x,q,y)	
{$C}@P{$D} {$C}^@P{$D}	PROP(a,p,b), C_SUB(c,a), C_SUB(d,b)	
{$C}@P {$C}^@P	PROP(a,p,b), C_SUB(c,a)	
@P{$D} ^@P{$D}	PROP(a,p,b), C_SUB(d,b)	
{X; ^$C}@P{Y; ^$D}	PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y), C_SUB(c,a), C_SUB(d,b), C_EXT(c,x), C_EXT(d,y)	RQLUCQ
{X; ^$C}@P{Y} {X; ^$C}@P	PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y), C_SUB(c,a), C_EXT(c,x)	
{X}@P{Y; ^$D} @P{Y; ^$D}	PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y), C_SUB(d,b), C_EXT(d,y)	
{X; $C}@P{Y; ^$D} {$C}@P{Y; ^$D}	PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y), C_SUB(c,a), C_SUB(d,b), C_SUB(e,c), C_EXT(e,x), C_EXT(d,y)	
{X; ^$C}@P{Y; $D} {X; ^$C}@P{$D}	PROP(a,p,b), P_SUB(q,p), P_EXT(x,q,y), C_SUB(c,a), C_SUB(d,b), C_SUB(f,d), C_EXT(c,x), C_EXT(f,y)	

{X; ^$C}^@P{Y; ^$D}	PROP(a,p,b), P_EXT(x,p,y), C_SUB(c,a), C_SUB(d,b), C_EXT(c,x), C_EXT(d,y)	
{X; ^$C}^@P{Y} {X; ^$C}^@P	PROP(a,p,b), P_EXT(x,p,y), C_SUB(c,a), C_EXT(c,x)	

{X}^@P{Y; ^$D}
^@P{Y; ^$D}
	PROP(a,p,b), P_EXT(x,p,y), C_SUB(d,b), C_EXT(d,y)	

{X; $C}^@P{Y; ^$D}
{$C}^@P{Y; ^$D}
	PROP(a,p,b), P_EXT(x,p,y), C_SUB(c,a), C_SUB(d,b), C_SUB(e,c), C_EXT(e,x), C_EXT(d,y)	

{X; ^$C}^@P{Y; $D}
{X; ^$C}^@P{$D}
	PROP(a,p,b), P_EXT(x,p,y), C_SUB(c,a), C_SUB(d,b), C_SUB(f,d), C_EXT(c,x), C_EXT(f,y)	

{X; $C}^@P{Y; $D}
{$C}^@P{Y; $D}
{X; $C}^@P{$D}
	PROP(a,p,b), P_EXT(x,p,y), C_SUB(c,a), C_SUB(d,b), C_SUB(e,c), C_SUB(f,d), C_EXT(e,x), C_EXT(f,y)	

{X; $C}^@P{Y} {$C}^@P{Y} {X; $C}^@P	PROP(a,p,b), P_EXT(x,p,y), C_SUB(c,a), C_SUB(e,c), C_EXT(e,x)	

{X}^@P{Y; $D}
{X}^@P{$D} ^@P{Y; $D}
	PROP(a,p,b), P_EXT(x,p,y), C_SUB(d,b), C_SUB(f,d), C_EXT(f,y)	

{X}^@P{Y}
{X}^@P ^@P{Y}
	P_EXT(x,p,y)	

Class Pattern	**SWLF Translation**	**Fragment**
$C ^$C	CLASS(c)	RQL_{CORE} when $C=c, $D=d RQL_{UCQ}
$C{$D} ^$C{$D}	C_SUB(d,c)	
$C{X}	C_SUB(d,c), C_EXT(d,x)	
$C{X; $D}	C_SUB(d,c), C_SUB(e,d), C_EXT(e,x)	
^$C{X}	C_EXT(d,x)	RQL_{UCQ}
^$C{X; $D}	C_SUB(d,c), C_SUB(e,d), C_EXT(e,x), C_EXT(c,x)	
$C{X; ^$D}	C_SUB(d,c), C_EXT(d,x)	
^$C{X; ^$D}	C_SUB(d,c), C_EXT(c,x), C_EXT(d,x)	