



Containment and Minimization of RDF/S Query Patterns

Giorgos Serfiotis, Ioanna Koffina

Computer Science Department, University of Crete
and Institute of Computer Science - FORTH

Val Tannen

Computer and Information Science Department, UPenn



Outline

- Motivation - Contribution
- From RDF/S to SWLF
- RDF/S Query Languages Fragments
- RQL Query Containment & Equivalence
- RQL Query Minimization
- Summary/Future Work



Why SQO for RDF/S Query Patterns?

- **Semantic Query Optimization (SQO)** has been proved useful in various applications:
 - **Integration of information sources** – query composition with views (mappings) introduces redundancies
 - **Automatic query production** (e.g., from graphical query generators) introduces redundancies
 - Exploitation of **cached query results**

- SQO has been extensively studied in the context of relational, deductive, object and XML databases
 - Less attention has been devoted in the context of the **Semantic Web**

I. Koffina



Motivation: Graphical Query Generation on SW Portals

The screenshot displays the ICS-FORTH Semantic Web Portal interface. At the top, it says "The ICS-FORTH Semantic Web Portal". Below this, there are two main sections for query generation:

- Classes Properties**: A tree view showing a hierarchy of classes and properties. The "creates" class is expanded, showing properties like first_name, exhibited, location, last_name, working_hours, and technique. The namespace is `http://139.91.183.10:9090/RDF/VRP/Examples/demo/culture.rdf#`.
- Classes Properties**: A tree view showing a hierarchy of classes and properties. The "ExtResource" class is expanded. The namespace is `http://139.91.183.10:9090/RDF/VRP/Examples/demo/admin.rdf#`.

On the right side, there is a query configuration area:





- Artist --> **creates** --> Artifact
- SubProperties:** sculpts paints
- Domain:** Artist
- Range:** Artifact

At the bottom right, there are buttons for "Eliminate duplicates", "Include all", "View query", and "Get Resources".

I. Koffina



Graphical Query Generation on SW Portals

Navigation	RQL Query	Path
creates Artist → Artifact 	<pre>select X₁, X₂ from {X₁}creates{X₂}</pre>	1 st : Artist → creates → Artifact
creates Artist → Artifact paints Painter → Painting   	<pre>select X₁, X₂ from {X₁}creates{X₂}, {X₁}paints{X₂}</pre> <p>Replaced By</p> <pre>select X₁, X₂ from {X₁}^paints{X₂}</pre>	2 nd : Painter → paints → Painting

I. Koffina



Contribution

- We study SQO for **expressive fragments of patterns** supported by the RDF/S query languages
 - We consider two fragments of increasing expressiveness allowing complex **pattern matching at the data**, but also, **at the schema level**
 - We adapt **sound and complete algorithms** proposed for relational conjunctive queries in order to check the containment of RDF/S query fragments

I. Koffina



Semantic Web Logic Framework (SWLF)

- We rely on:
 - union of conjunctive queries of the form:

$$\bigcup_{i=1}^l q(\vec{x}) \leftarrow \varphi_i(\vec{x}, \vec{y}_i)$$

- and constraints under the form of disjunctive embedded dependencies (DEDs):

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

- Our first-order logic framework consists of:
 - a set of six FOL predicates capturing RDF/S schemas and description bases
 - and a set of FOL constraints capturing RDF semantics

I. Koffina



Semantic Web Logic Framework (SWLF)

- Several reasons for adopting a first-order logic (FOL) framework along with constraints:
 - By reducing the SQO problem for RDF/S query patterns to a relational equivalent, we exploit background theory on relational query optimization
 - query containment (equivalence),
 - query minimization
 are solvable for a fairly large class of queries in the presence of certain classes of constraints
 - A robust formalism to rely on: union of conjunctive queries of the form:

$$\bigvee_{i=1}^l q(\vec{x}) \leftarrow \varphi_i(\vec{x}, \vec{y}_i)$$

I. Koffina



SWIM Logic Framework (SWLF)

- The **constraints** considered in SWLF are **disjunctive embedded dependencies** (DEDs) of the form:

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

- Our first-order logic framework consists of:
 - A set of **six FOL predicates** capturing RDF/S schemas and description bases
 - and a set of **FOL constraints** (DEDs)

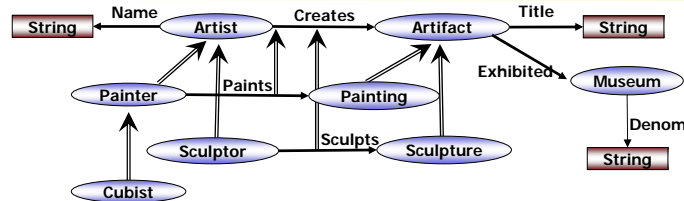


RDF/S Representation in SWLF

- **RDF/S schemas and resource descriptions** are represented by the following predicates:
 - **CLASS(name:Class)**: All the RDF/S schema classes
 - **C_SUB(subC:Class, class:Class)**: Class *subC* is a subclass of *class*
 - **PROP(subject:Class, predicate:Property, object:Class)**: All the RDF/S schema properties (*predicate*) with domain (*subject*) and range (*object*) restrictions
 - **P_SUB(subP:Property, prop:Property)**: Property *subP* is a subproperty of *prop*
 - **C_EXT(class:Class, inst:Resource)**: Resource *inst* belongs to the proper extent of *class*
 - **P_EXT(subject:Resource, predicate:Property, object:Resource)**: Resources *subject* and *object* belong to the proper extent of property *predicate*



From RDF/S Schemas to SWLF: Example



Some facts for the RDF/S schema above:

CLASS

name	subject	predicate	object
Artist	Artist	Name	String
Artifact	Artifact	Title	String
Painter	Artist	Creates	Artifact
Painting	Painter	Paints	Painting

C_SUB

subC	class
Painter	Artist
Painting	Artifact

P_SUB

subP	prop
Sculpts	Creates
Paints	Creates

C_EXT(class, instance)

and

P_EXT(subject, predicate, object)

I. Koffina



Constraints Capturing Schema Information

- For each SWLF predicate φ modelling RDF/S schemas we extract
 - one constraint **universally quantifying** the predicate's variables

$$\forall x \left[\varphi(x) \rightarrow \bigvee_{i=1}^l \varphi'_i(x) \right] \quad (\text{I})$$

Example: For the P_SUB predicate this constraint is:

$$\forall \text{subP} \forall \text{prop} (P_SUB(\text{subP}, \text{prop}) \rightarrow (\text{subP} = \text{"Paints"} \wedge \text{prop} = \text{"Creates"}) \vee (\text{subP} = \text{"Sculpts"} \wedge \text{prop} = \text{"Creates"}) \vee (\text{subP} = \text{"Paints"} \wedge \text{prop} = \text{"Paints"}) \vee (\text{subP} = \text{"Sculpts"} \wedge \text{prop} = \text{"Sculpts"}) \vee (\text{subP} = \text{"Creates"} \wedge \text{prop} = \text{"Creates"}))$$

- many constraints **existentially quantifying** them

$$\exists x \varphi(x) \quad (\text{II})$$

which is equivalent to $\varphi(x)$

Example:

$$\exists b \exists d P_SUB(d, b) \wedge d = \text{"Paints"} \wedge b = \text{"Creates"} \text{ OR } P_SUB(\text{Paints}, \text{Creates})$$

$$\exists b \exists d P_SUB(d, b) \wedge d = \text{"Sculpts"} \wedge b = \text{"Creates"} \text{ OR } P_SUB(\text{Sculpts}, \text{Creates})$$

General RDF/S Constraints: δ_{Mod}

Basic constraints:

- every resource in the extent of a class implies the existence of the corresponding class

$$\forall c, x [C_EXT(c, x) \rightarrow CLASS(c)]$$

- every statement in the extent of a property implies the existence of the corresponding property

$$\forall x, p, y [P_EXT(x, p, y) \rightarrow \exists c, d PROP(c, p, d)]$$

- every subclass of a class is also a class

$$\forall c_1, c_2 [C_SUB(c_1, c_2) \rightarrow CLASS(c_1) \wedge CLASS(c_2)]$$

- every sub-property of a property is also a property

$$\forall p, q [P_SUB(p, q) \rightarrow \exists a, b, c, d [PROP(a, p, b) \wedge PROP(c, q, d)]]$$

- the domain & range of every property is a class

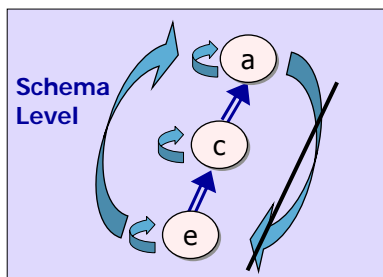
$$\forall c_1, c_2, p [PROP(c_1, p, c_2) \rightarrow CLASS(c_1) \wedge CLASS(c_2)]$$

- the domain & range of every property is unique

$$\forall a_1, a_2, p, b_1, b_2 [PROP(a_1, p, b_1) \wedge PROP(a_2, p, b_2) \rightarrow a_1 = a_2 \wedge b_1 = b_2]$$

I. Koffina

General RDF/S Constraints: δ_{Mod}



Every class is a subclass of itself and the subclass relationship is **transitive**. There can be **no cycle** in the class hierarchy

SUB constraints

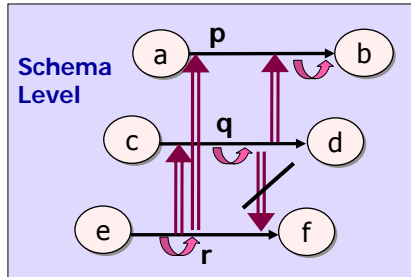
- $\forall c (CLASS(c) \rightarrow C_SUB(c, c))$ (subclass reflexivity)

- $\forall c_1, c_2, c_3 (C_SUB(c_1, c_2) \wedge C_SUB(c_2, c_3) \rightarrow C_SUB(c_1, c_3))$ (subclass transitivity)

- $\forall c_1, c_2 (C_SUB(c_1, c_2) \wedge C_SUB(c_2, c_1) \rightarrow c_1 = c_2)$ (subclass antisymmetry)

I. Koffina

General RDF/S Constraints: δ_{Mod}

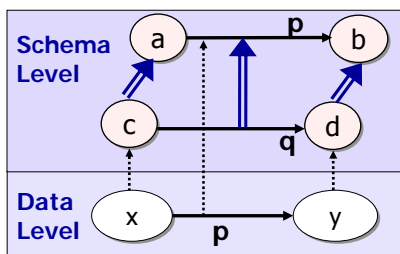


Every property is a subproperty of itself and the subproperty relationship is transitive. There can be no cycle in the property hierarchy

- ④ $\forall a, p, b (PROP(a, p, b) \rightarrow P_SUB(p, p))$ (subproperty reflexivity)
- ⑤ $\forall p_1, p_2, p_3 (P_SUB(p_1, p_2) \wedge P_SUB(p_2, p_3) \rightarrow P_SUB(p_1, p_3))$ (subproperty transitivity)
- ⑥ $\forall p_1, p_2 (P_SUB(p_1, p_2) \wedge P_SUB(p_2, p_1) \rightarrow p_1 = p_2)$ (subproperty antisymmetry)

I. Koffina

General RDF/S Constraints: δ_{Mod}



In a valid RDF description schema the domain (range) of every sub-property is subsumed by the domain (range) of its super-property

In a valid RDF description base the subject/object resources in every statement are instances of the property's domain/range classes (either direct or indirect instances)

➤ Domain-range constraints

- ① $\forall a, p, b, c, q, d [P_SUB(q, p) \wedge PROP(a, p, b) \wedge PROP(c, q, d) \rightarrow C_SUB(c, a) \wedge C_SUB(d, b)]$ (subproperty-subclass compatibility)
- ② $\forall a, p, b, x, y [PROP(a, p, b) \wedge P_EXT(x, p, y) \rightarrow \exists c, d [C_SUB(c, a) \wedge C_SUB(d, b) \wedge C_EXT(c, x) \wedge C_EXT(d, y)]]$ (property-class extent compatibility)

I. Koffina



SWLF Semantics vs. RDF-MT

- RDF-MT does not consider different sorts of resources
 - SWLF distinguish between classes, properties and their corresponding instances
- Additional SWLF Constraints
 - SWLF 6th basic constraint stating that each property has exactly one domain & range
 - SWLF 3rd & 6th sub constraints stating that there can be no cycle neither in the class nor in the property hierarchy
 - SWLF 1st domain-range constraint stating that the domain (range) of every sub-property is subsumed by the domain (range) of its super-property
 - SWLF 2nd domain-range constraint stating the subject and object resources are (direct or not) instances of the domain and range classes of the property, respectively

I. Koffina



Constraints in SWLF

- Sets of constraints considered in SWLF
 - δ_{Mod} : The set of general RDF/S constraints stating and preserving RDF/S semantics
 - δ_{RDF} : δ_{Mod} along with the constraints of form (II)
 - Δ_{RDF} : δ_{Mod} along with the constraints of forms (I) and (II)

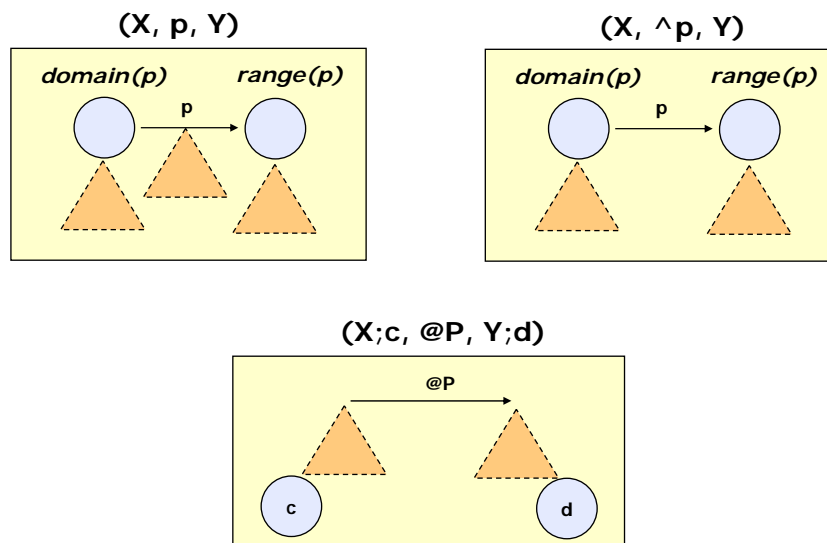
I. Koffina

RDF/S Query Languages Fragments

- RDF/S query languages provide **pattern matching** facilities against **schema and/or data graphs**
- 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 can be divided into two main fragments:
 - includes patterns for **pure data matching** given as input the schema classes and properties (i.e., as in relational queries)
 - includes patterns that **arbitrary mix schema and data querying**
 - When both exact and extended matching is supported, patterns of **schema-aware** RDF/S query languages can be similarly divided into the previous two fragments while the latter also includes patterns for exact matching of schema and data graphs.

I. Koffina

RDF/S Query Patterns



I. Koffina



RDF/S Query Patterns Categorization

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

- RQL supports **generalized path expressions** featuring variables on labels for both nodes (i.e., classes) and edges (i.e., properties)
- It allows **querying both schema and/or data**

I. Koffina



RQL Query Fragments

- An **RQL conjunctive query** is a FOL formula of the form:

$$ans(\bar{U}) :- \dots, E_i(\bar{U}_i), \dots, u_m = u_n, \dots$$

- \bar{U} is a tuple of variables or constants
- $E_i(\bar{U}_i)$ are class/property patterns and $u_m = u_n$ are equalities between variables and/or constants
- \bar{U}_i 's involve 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
- RQL_{UCQ} (RQL_{CORE}) queries have the form: $\bigcup_k Q_k$
where Q_k are **conjunctive queries whose heads have the same type**

I. Koffina



From RDF/S Patterns to SWLF

- Some **class patterns** and their translation:

$\$C\{X\}$: C_SUB(d, c), C_EXT(d, x)

$\wedge\$C\{X\}$: C_EXT(c, x)

- Some **property patterns** and their translation:

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

$\{X; \wedge\$C\}^\wedge@P\{Y; \wedge\$D\}$: 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)

- There are ten class patterns and fifty-one (!) property patterns



From RQL Queries to SWLF

- Translation of the RQL_{UCQ} query retrieving cubists who have painted artifacts 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:

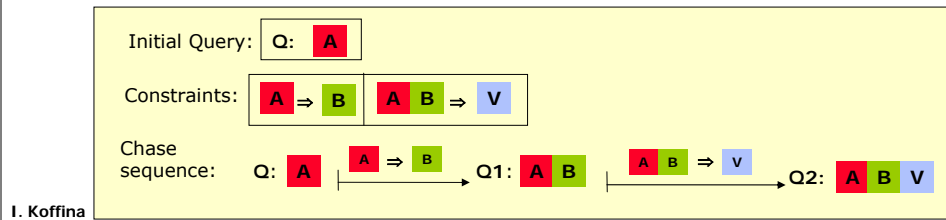
```
select X
from {X; $C}@P1{Y}, {Y}@P2
where $C=Cubist and @P1=Paints and @P2=Exhibited
```

Now the query can be rewritten as follows:

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

Chase Algorithm

- Chasing a query corresponds to applying a **sequence of chase steps**
- A chase step applies if both:
 - There is a mapping (**homomorphism**) from a dependency's universally quantified variables to the query's variables
 - There is no homomorphism from the dependency's conclusion into the query's body (to avoid endless chase steps)
- Each query is chased with **all available dependencies** until **no more chase steps** apply
- The resulting query is called the **universal plan**



Query Containment

- **Theorem:** Suppose two unions of conjunctive queries Q_1 and Q_2 . Given that the chase of Q_1 with a set Δ of constraints terminates yielding the universal plan U_1 , Q_1 is contained in Q_2 under Δ iff for every i there is a j such that there is a **containment mapping** from Q_{2j} into U_{1i}
- The containment problem for RQL_{CORE} queries corresponds to the above one
 - δ_{RDF} instead of Δ



RQL_{CORE} Query Containment

- Let's check the containment of RQL_{CORE} query

SELECT X	in	SELECT X
FROM Painter{X}		FROM Artist{X}

The first one translates to $(Q_1) \text{ ans}(x) :- C_SUB(c, a), C_EXT(c, x), a=\text{"Painter"}$

while the second one to $(Q_2) \text{ ans}(x) :- C_SUB(c, a), C_EXT(c, x), a=\text{"Artist"}$

The query Q_1 chases with constraint $\exists b \exists d C_SUB(d, b) \wedge d=\text{"Painter"} \wedge b=\text{"Artist"}$

and then with constraint $\forall c_1 \forall c_2 \forall c_3 C_SUB(c_1, c_2) \wedge C_SUB(c_2, c_3) \rightarrow C_SUB(c_1, c_3)$ resulting in query

$(U_1) \text{ ans}(x) :- C_SUB(c, a), C_SUB(d, b), C_SUB(c, b), C_EXT(c, x), a=\text{"Painter"},$ $d=\text{"Painter"}, b=\text{"Artist"}$
--

There is a containment mapping from Q_2 to U_1 using the homomorphism $\{x \rightarrow x, c \rightarrow c, a \rightarrow b\}$. Thus, Q_1 is contained in Q_2 under Δ (δ_{RDF})

I. Koffina



RQL_{UCQ} Query Containment

- Likewise, for the RQL_{UCQ} query containment problem
 - Δ_{RDF} instead of set Δ

- By chasing with Δ_{RDF} we can show that

SELECT X, Y	$(\text{ans}(x,y) :- P_SUB(q, p), P_EXT(x, q, y), p=\text{"Paints"})$
FROM {X}Paints{Y}	

is contained in

SELECT X, Y	$(\text{ans}(x,y) :- P_EXT(x, p, y), p=\text{"Paints"})$
FROM {X}^Paints{Y}	

I. Koffina



RQL_{UCQ} (RQL_{CORE}) Query Equivalence

- **Theorem:** Two unions of conjunctive queries Q_1 and Q_2 are equivalent ($Q_1 \equiv_{\Delta} Q_2$) under a set Δ of constraints iff $Q_1 \subseteq_{\Delta} Q_2$ and $Q_2 \subseteq_{\Delta} Q_1$
 - Therefore, for the RQL_{UCQ} and RQL_{CORE} query equivalence problems, we employ the theorem above using Δ_{RDF} and δ_{RDF} , respectively
- Obviously, the RQL_{UCQ} (RQL_{CORE}) equivalence check corresponds to **minimum one** and **maximum two** RQL_{UCQ} (RQL_{CORE}) containment checks



Termination and Complexity of Chase

- The **termination of chase is not guaranteed** for sets of arbitrary DEDs
- However, δ_{RDF} and Δ_{RDF} guarantee termination: they satisfy the **stratified-witness** property
- Stratified-witness ensures that **infinite fresh variables will not be introduced** during chase and that the latter **will not diverge**
- **Proposition:** The chase of any query Q with any set of constraints Δ **with stratified-witness terminates**. If Q is conjunctive and the constraints are disjunction-free, the chase is **NP-complete**
- **Universal plan's size:** $O(|Q|^{d+1})$



Termination and Complexity of Chase

- The **termination of chase is not guaranteed** for sets of arbitrary DEDs
- However, δ_{RDF} and Δ_{RDF} guarantee termination: they satisfy the **stratified-witness** property
- Stratified-witness ensures that **infinite fresh variables will not be introduced** during chase and that the latter **will not diverge**
- **Proposition:** The **chase** of any query Q with any set of constraints Δ **with stratified-witness terminates**. If Q is conjunctive and the constraints are disjunction-free, the chase is **NP-complete**



Query Minimization: Backchase

- In order to minimize a query Q , given its universal plan, we apply the **backchase** algorithm
 - In the backchase phase, all **universal plan's subqueries are checked for equivalence with the query** using the chase
 - Exponential number of NP-complete problems when chase ends
- A subquery is induced by a subset of the atoms in the body of the universal query
 - **All Δ -minimal queries Δ -equivalent to the original one can be found this way**
- A query is **Δ -minimal** if **no relational atoms can be removed** from it without compromising the Δ -equivalence to it
- Intuitively, a minimal query uses **fewer** and/or **simpler** RDF/S query patterns than the original one



RQL_{CORE} (RQL_{UCQ}) Query Minimization

- Suppose the RQL_{CORE} query

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

It translates to

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

After chasing the query, we examine the subquery

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

It is equivalent to the original one under δ_{RDF} and no relational atom can be removed from it. Therefore, it is minimal.

- The same query, if considered as an RQL_{UCQ} one, minimizes under Δ_{RDF} to the query

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

I. Koffina



RQL_{UCQ} Query Minimization: An Interesting Example

- Assume the RQL_{UCQ} query

```
Q:  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"
```

It has three minimal equivalents under Δ_{RDF} :

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

I. Koffina



RQL_{UCQ} Minimization by Ignoring Schema Information

- The chase in this case considers only δ_{Mod}
- 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)
```

It minimizes to

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

which corresponds to the RQL_{UCQ} query

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



Backward Translation

- Simple for RQL_{CORE} queries
 - Only $\$C\{X\}$ and $\{X\}@P\{Y\}$ may appear
- More complex for RQL_{UCQ}
 - Initially, we identify simple patterns
 - Then, combine them into more complex ones
- For both RQL_{UCQ} and RQL_{CORE} minimal queries we reduce the number of employed variables and replace with constants as many variables as possible by using the equalities between the variables and constants

In the 1st phase, the first minimal query of the previous example translates into:

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

The 2nd phase does not affect it. After the 3rd phase we get the RQL_{UCQ} query

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



Summary & Future Work

- This work relies on background knowledge from **relational database theory**
 - RDF/S Data Model and Semantics represented by
 - **first-order** (relational) **predicates**, and
 - **a set of constraints** (DEDs)
 - **Chase & Backchase** algorithms (sound & complete)
 - Check for containment/equivalence and minimization of RDF/S queries
- Future extensions
 - Study SQO of constructs originating from **more expressive** SW ontology languages, such as OWL's **inverse properties** as well as **disjointness of class and property** interpretations
 - Study SQO of **richer RDF/S query fragments** involving **functions** – like domain, range, subclassof, subpropertyof and aggregate ones – as well as **nested queries**



Questions ???