Expressive Languages for Querying the Semantic Web

Marcelo Arenas\(^1\) \hspace{1cm} Georg Gottlob\(^2\) \hspace{1cm} Andreas Pieris\(^2\)

\(^1\)Department of Computer Science, PUC Chile, Chile
\(^2\)Department Computer Science, University of Oxford, UK

Dagstuhl 2014
Resource Description Framework (RDF)

- **Data model** for representing information about web resources

- **Uniform Resource Identifier (URI)** - http://dbpedia.org/resource/Jeffrey_Ullman

- URIs are organized as **RDF graphs** - \((subject, predicate, object)\)
Resource Description Framework (RDF)

- **Data model** for representing information about web resources

- **Uniform Resource Identifier** (URI) - http://dbpedia.org/resource/Jeffrey_Ullman

- URIs are organized as **RDF graphs** - (subject, predicate, object)

![RDF Graph Example](http://dbpedia.org/resource/)
Resource Description Framework (RDF)

- **Data model** for representing information about web resources

- **Uniform Resource Identifier (URI)** - http://dbpedia.org/resource/Jeffrey_Ullman

- URIs are organized as **RDF graphs** - \((\text{subject}, \text{predicate}, \text{object})\)

  (dbpedia:Ullman, name, “Jeffrey Ullman”)
  (dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)
  (dbpedia:Aho, name, “Alfred Aho”)

  dbpedia: <http://dbpedia.org/resource/>
SPARQL

- Graph-matching query language
- First public working draft in 2004 by W3C
- W3C recommendation in 2008
SPARQL

- Graph-matching query language
- First public working draft in 2004 by W3C
- W3C recommendation in 2008

```
SELECT ?X
WHERE {
  ?Y  is_author_of  ?Z .
  ?Y  name  ?X .}
```
SPARQL

- Graph-matching query language
- First public working draft in 2004 by W3C
- W3C recommendation in 2008

```
SELECT ?X

(?Y, is_author_of, ?Z) AND (?Y, name, ?X)
```

algebraic syntax introduced in [Pérez, A. & Gutierrez, TODS 2009]
SELECT ?X

(?Y, is_author_of, ?Z) AND (?Y, name, ?X)

(dbpedia:Ullman, name, “Jeffrey Ullman”)
(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)
(dbpedia:Aho, name, “Alfred Aho”)
SELECT ?X

(??Y, is_author_of, ?Z) AND (??Y, name, ?X)

(dbpedia:Ullman, name, “Jeffrey Ullman”)
(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)
(dbpedia:Aho, name, “Alfred Aho”)

Answer: “Jeffrey Ullman”
RDFS and OWL Vocabularies

(dbpedia:Ullman, name, “Jeffrey Ullman”)
(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)
(dbpedia:Aho, name, “Alfred Aho”)

(r₁, rdf:type, owl:Restriction)
(r₁, owl:onProperty, is_coauthor_of)
(r₁, owl:someValuesFrom, owl:Thing)

(r₂, rdf:type, owl:Restriction)
(r₂, owl:onProperty, is_author_of)
(r₂, owl:someValuesFrom, owl:Thing)

(r₁, rdfs:subClassOf, r₂)
RDFS and OWL Vocabularies

(dbpedia:Ullman, name, “Jeffrey Ullman”)
(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)
(dbpedia:Aho, name, “Alfred Aho”)

(r₁, rdf:type, owl:Restriction)
(r₁, owl:onProperty, is_coauthor_of)
(r₁, owl:someValuesFrom, owl:Thing)

(r₂, rdf:type, owl:Restriction)
(r₂, owl:onProperty, is_author_of)
(r₂, owl:someValuesFrom, owl:Thing)

(r₁, rdfs:subClassOf, r₂)

\[ r₁ = \{ a \mid \text{there exists a URI } b \text{ such that } (a, \text{is_coauthor_of}, b) \} \]

\[ r₂ = \{ a \mid \text{there exists a URI } b \text{ such that } (a, \text{is_author_of}, b) \} \]
RDFS and OWL Vocabularies

(dbpedia:Ullman, name, “Jeffrey Ullman”)  
(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)  
(dbpedia:Aho, name, “Alfred Aho”)  

(r₁, rdf:type, owl:Restriction)  
(r₁, owl:onProperty, is_coauthor_of)  
(r₁, owl:someValuesFrom, owl:Thing)  

(r₂, rdf:type, owl:Restriction)  
(r₂, owl:onProperty, is_author_of)  
(r₂, owl:someValuesFrom, owl:Thing)  

(r₁, rdfs:subClassOf, r₂)  

each co-author is an author
RDFS and OWL Vocabularies

(dbpedia:Ullman, name, “Jeffrey Ullman”)
(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)
(dbpedia:Aho, name, “Alfred Aho”)

SELECT ?X

(?Y, is_author_of, ?Z) AND (?Y, name, ?X)

Expected answers: Jeffrey Ullman
Alfred Aho
RDFS and OWL Vocabularies

(dbpedia:Ullman, name, “Jeffrey Ullman”)
(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)
(dbpedia:Aho, name, “Alfred Aho”)

(r₁, rdf:type, owl:Restriction)
(r₁, owl:onProperty, is_coauthor_of)
(r₁, owl:someValuesFrom, owl:Thing)

(r₂, rdf:type, owl:Restriction)
(r₂, owl:onProperty, is_author_of)
(r₂, owl:someValuesFrom, owl:Thing)

(r₁, rdfs:subClassOf, r₂)

SELECT ?X

(?Y, is_author_of, ?Z) AND (?Y, name, ?X)

Expected answers: Jeffrey Ullman ✓
Alfred Aho ✗
RDFS and OWL Vocabularies

(dbpedia:Ullman, name, “Jeffrey Ullman”)
(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)
(dbpedia:Aho, name, “Alfred Aho”)

(r1, rdf:type, owl:Restriction)
(r1, owl:onProperty, is_coauthor_of)
(r1, owl:someValuesFrom, owl:Thing)

(r2, rdf:type, owl:Restriction)
(r2, owl:onProperty, is_author_of)
(r2, owl:someValuesFrom, owl:Thing)
(r1, rdfs:subClassOf, r2)

SELECT ?X
(?Y, rdf:type, ?Z) AND
(?Z, rdf:type, owl:Restriction) AND
(?Z, owl:onProperty, is_author_of) AND
(?Z, owl:someValuesFrom, owl:Thing)
AND (?Y, name, ?X)

Jeffrey Ullman
Alfred Aho
RDFS and OWL Vocabularies

(dbpedia:Ullman, name, “Jeffrey Ullman”)
(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)
(dbpedia:Aho, name, “Alfred Aho”)

(r₁, rdf:type, owl:Restriction)
(r₁, owl:onProperty, is_coauthor_of)
(r₁, owl:someValuesFrom, owl:Thing)

(r₂, rdf:type, owl:Restriction)
(r₂, owl:onProperty, is_author_of)
(r₂, owl:someValuesFrom, owl:Thing)
(r₁, rdfs:subClassOf, r₂)

SELECT ?X

(?Y, rdf:type, ?Z) AND
(?Z, rdf:type, owl:Restriction) AND
(?Z, owl:onProperty, is_author_of) AND
(?Z, owl:someValuesFrom, owl:Thing)

AND (?Y, name, ?X)

Jeffrey Ullman ✓
Alfred Aho ✓
Need for Decoupling

SELECT ?X
    (?Y, is_author_of, ?Z) AND (?Y, name, ?X)

vs

SELECT ?X
    (?Y, rdf:type, ?Z) AND
    (?Z, rdf:type, owl:Restriction) AND
    (?Z, owl:onProperty, is_author_of) AND
    (?Z, owl:someValuesFrom, owl:Thing)
    AND (?Y, name, ?X)
Our Objectives

• Decouple the reasoning part and the actual query - simpler queries

RDF graph → RDFS/OWL vocabulary → actual query

fixed and query independent
Our Objectives

• **Decouple** the reasoning part and the actual query - simpler queries

  RDF graph → RDFS/OWL vocabulary → actual query

  fixed and query independent

• **Navigational capabilities** - exploit the graph structure of RDF data

• **General form of recursion** - central feature for graph query languages
The rest of the Talk

- The modular query language TriQ
- From SPARQL over OWL 2 QL to TriQ
- TriQ-Lite - a tractable language
- Concluding remarks
Triple Query Language (TriQ)

\[ M = [Q_{RDFS/OWL}, Q_{SPARQL}] \]

\[ ans(M,G) = ans(Q_{SPARQL}, ans(Q_{RDFS/OWL}, G)) \]
What is the right syntax for $Q_{RDFS/OWL}$ and $Q_{SPARQL}$?
Triple Query Language (TriQ)

\[ M = [Q_{RDFS/OWL}, Q_{SPARQL}] \]

\[ ans(M,G) = ans(Q_{SPARQL}, ans(Q_{RDFS/OWL}, G)) \]

What is the right syntax for \( Q_{RDFS/OWL} \) and \( Q_{SPARQL} \)?

- Datalog[¬s] represents every SPARQL query
- Datalog[∃, ⊥] is appropriate for ontological reasoning

Datalog[∃, ¬s, ⊥]
Triple Query Language (TriQ)

\[ M = [Q_{\text{RDFS/OWL}}, Q_{\text{SPARQL}}] \]

\[ \text{ans}(M, G) = \text{ans}(Q_{\text{SPARQL}}, \text{ans}(Q_{\text{RDFS/OWL}}, G)) \]

What is the right syntax for \( Q_{\text{RDFS/OWL}} \) and \( Q_{\text{SPARQL}} \)?

- Datalog\( [\neg s] \) represents every SPARQL query
- Datalog\( [\exists, \bot] \) is appropriate for ontological reasoning

\[ \text{weakly-guarded Datalog}[\exists, \neg s, \bot] \]
Weakly-Guarded Datalog\[\exists, \neg s\]

All body-variables at **affected positions** occur in a positive body-atom

appears in an \(\exists\)-position or just in affected positions in the body

\[
P(\textit{?X}, \textit{?Y}), S(\textit{?Y}, \textit{?Z}) \rightarrow \exists\textit{?W} \ T(\textit{?Y}, \textit{?X}, \textit{?W})
\]

\[
T(\textit{?X}, \textit{?Y}, \textit{?Z}) \rightarrow \exists\textit{?W} \ P(\textit{?W}, \textit{?Y})
\]

\[
P(\textit{?X}, \textit{?Y}), \neg R(\textit{?X}) \rightarrow \exists\textit{?Z} \ Q(\textit{?X}, \textit{?Z})
\]

Affected positions = ?
Weakly-Guarded Datalog[∃,¬s]

All body-variables at affected positions occur in a positive body-atom
appears in an ∃-position or just in affected positions in the body

\[ T(?X, ?Y, ?Z) \rightarrow ∃?W \ P(?W, ?Y) \]
\[ P(?X, ?Y), ¬R(?X) \rightarrow ∃?Z \ Q(?X, ?Z) \]

Affected positions = \{T[3], P[1], Q[2]\}
Weakly-Guarded Datalog[$\exists, \neg s$]

All body-variables at affected positions occur in a positive body-atom

appears in an $\exists$-position or just in affected positions in the body

\[
\begin{align*}
T(?X, ?Y, ?Z) & \rightarrow \exists ?W \ P(?W, ?Y) \\
P(?X, ?Y), \neg R(?X) & \rightarrow \exists ?Z \ Q(?X, ?Z)
\end{align*}
\]

Affected positions = \{T[3], P[1], Q[2], T[2]\}
Weakly-Guarded Datalog[$\exists, \neg s$]

All body-variables at **affected positions** occur in a positive body-atom

appears in an $\exists$-position or just in affected positions

in the body

\[ P(?X, ?Y), S(?Y, ?Z) \rightarrow \exists?W \ T(?Y, ?X, ?W) \]

\[ T(?X, ?Y, ?Z) \rightarrow \exists?W \ P(?W, ?Y) \]

\[ P(?X, ?Y), \neg R(?X) \rightarrow \exists?Z \ Q(?X, ?Z) \]

Affected positions = \{ T[3], P[1], Q[2], T[2], P[2] \}
Weakly-Guarded Datalog[$\exists,\neg s$]

All body-variables at affected positions occur in a positive body-atom

appears in an $\exists$-position or just in affected positions in the body

\[
\]
\[
\]
\[
P(?X, ?Y), \neg R(?X) \rightarrow \exists?Z \ Q(?X, ?Z)
\]

Affected positions = \{T[3], P[1], Q[2], T[2], P[2], Q[1], \}
Weakly-Guarded Datalog[$\exists, \neg s$]

All body-variables at affected positions occur in a positive body-atom

appears in an $\exists$-position or just in affected positions in the body

\[
\]
\[
\]
\[
P(?X, ?Y), \neg R(?X) \rightarrow \exists?Z \ Q(?X, ?Z)
\]

Affected positions = \{T[3], P[1], Q[2], T[2], P[2], Q[1],\}
Weakly-Guarded Datalog[∃,¬s]

All body-variables at affected positions occur in a positive body-atom appears in an ∃-position or just in affected positions in the body

\[
\begin{align*}
P(?X, ?Y), S(?X, ?Z) & \rightarrow \exists W \ T(?Y, ?X, ?W) \\
T(?X, ?Y, ?Z) & \rightarrow \exists W \ P(?W, ?Y) \\
P(?X, ?Y), \neg R(?X) & \rightarrow \exists Z \ Q(?X, ?Z)
\end{align*}
\]

Affected positions = \{T[3], P[1], Q[2], T[2], P[2], Q[1] \}
Weakly-Guarded Datalog[∃,¬s]

All body-variables at affected positions occur in a positive body-atom

appears in an ∃-position or just in affected positions in the body


\[ T(?X, ?Y, ?Z) \rightarrow ∃?W \ P(?W, ?Y) \]

\[ P(?X, ?Y), \neg R(?X) \rightarrow ∃?Z \ Q(?X, ?Z) \]

Affected positions = \{ T[3], P[1], Q[2], T[2], P[2], Q[1] \}
Weakly-Guarded Datalog[$\exists, \neg s, \bot$]

All body-variables at affected positions occur in a positive body-atom appears in an $\exists$-position or just in affected positions in the body

weakly-guarded Datalog[$\exists, \neg s$] + $\Phi(x_1, \ldots, x_k) \rightarrow \bot$

= weakly-guarded Datalog[$\exists, \neg s, \bot$]
Triple Query Language (TriQ)

\[ M = [Q_{\text{RDFS/OWL}}, Q_{\text{SPARQL}}] \]

weakly-guarded Datalog[∃, ¬, ⊥] queries
Triple Query Language (TriQ)

\[ M = [Q_{RDFS/OWL}, Q_{SPARQL}] \]

weakly-guarded Datalog[∃, ¬s, ⊥] queries

Weakly-guarded Datalog[∃, ¬s, ⊥] query: (Π, Λ)

- Π is a weakly-guarded Datalog[∃, ¬s, ⊥] program
- Λ is a set of answer rules: \( \Phi(x_1, ..., x_k) \rightarrow answer(x_1, ..., x_k) \)
Triple Query Language (TriQ): Complexity

- **Theorem**: Query evaluation for TriQ is in EXPTIME
Triple Query Language (TriQ): Complexity

- **Theorem**: Query evaluation for TriQ is in EXPTIME

- **Theorem** [Gottlob, Rudolph & Šimkus, PODS 2014]: Every query that can be evaluated in EXPTIME can be expressed in weakly-guarded Datalog[∃,¬s] (no order)
Triple Query Language (TriQ): Complexity

- **Theorem**: Query evaluation for TriQ is in EXPTIME

- **Theorem** [Gottlob, Rudolph & Šimkus, PODS 2014]: Every query that can be evaluated in EXPTIME can be expressed in weakly-guarded Datalog[∃,¬s] (no order)

- **Corollary**: TriQ and weakly-guarded Datalog[∃,¬s] capture EXPTIME (no order)
For every object $a$, we ask for the name and the phone of $a$, if the phone number of $a$ is available; otherwise, we only ask for the name of $a$.

$$Q = (\exists X, \text{name}, ?Y) \text{ OPT } (\exists X, \text{phone}, ?Z)$$

$A \quad B$
From SPARQL to TriQ

\[ Q = (\text{?X, name, ?Y}) \text{ OPT (?X, phone, ?Z)} \]

for every object \( a \), we ask for the name and the phone of \( a \), if the phone number of \( a \) is available; otherwise, we only ask for the name of \( a \)

\[ M_Q = [ (\emptyset, \text{\( t_{bgp}(Q) \))}, (\text{\( t_{opr}(Q) \)}, \text{\( t_{out}(Q) \)}) ] \]

- evaluate basic graph patterns (\( A \) and \( B \))
- output rules
- encode the semantics of SPARQL operators (OPT)
From SPARQL to TriQ

\[
Q = (?X, \text{name}, ?Y) \ \text{OPT} \ (?X, \text{phone}, ?Z)
\]

\[
A \quad B
\]

The program \( \tau_{\text{bgp}}(Q) \):

\[
\begin{align*}
\text{triple}(?X, \text{name}, ?Y) & \rightarrow \ \text{query}_A(?X, ?Y) \\
\text{triple}(?X, \text{phone}, ?Y) & \rightarrow \ \text{query}_B(?X, ?Y)
\end{align*}
\]
From SPARQL to TriQ

\[ Q = (\?X, \text{name, ?Y}) \text{ OPT (\?X, \text{phone, ?Z})} \]

The program \( \tau_{opr}(Q) \):

\[ \text{query}_A(\?X, \?Y), \text{query}_B(\?X, \?Z) \rightarrow \text{query}_Q(\?X, \?Y, \?Z) \]

list of individuals with phone number

\[ \text{query}_A(\?X, \?Y), \text{query}_B(\?X, \?Z) \rightarrow \text{compatible}_Q(\?X) \]
From SPARQL to TriQ

\[ Q = (?X, \text{name}, ?Y) \quad \text{OPT} \quad (?X, \text{phone}, ?Z) \]

The program \( \tau_{\text{opr}}(Q) \):

\[
\begin{align*}
\text{query}_A(?X, ?Y), \text{query}_B(?X, ?Z) & \rightarrow \text{query}_Q(?X, ?Y, ?Z) \\
\text{list of individuals with phone number} \\
\text{query}_A(?X, ?Y), \text{query}_B(?X, ?Z) & \rightarrow \text{compatible}_Q(?X) \\
\text{the third argument (phone number) is missing} \\
\text{query}_A(?X, ?Y), \neg\text{compatible}_Q(?X) & \rightarrow \text{query}_{Q,(3)}(?X, ?Y)
\end{align*}
\]
From SPARQL to TriQ

\[ Q = (\?X, \text{name}, \?Y) \quad \text{OPT} \quad (\?X, \text{phone}, \?Z) \]

\[ A \quad B \]

The program \( \tau_{out}(Q) \):

\[
\text{query}_Q(\?X, \?Y, \?Z) \rightarrow \text{answer}_Q(\?X, \?Y, \?Z)
\]

\[
\text{query}_{Q,\{3\}}(\?X, \?Y) \rightarrow \text{answer}_{Q,\{3\}}(\?X, \?Y)
\]
From SPARQL to TriQ

\[ Q = (?X, \text{name}, ?Y) \text{ OPT } (?X, \text{phone}, ?Z) \]

A

B

The program \( \tau_{\text{out}}(Q) \):

\[ \text{query}_{Q,\emptyset}(?X, ?Y, ?Z) \rightarrow \text{answer}_{Q,\emptyset}(?X, ?Y, ?Z) \]

\[ \text{query}_{Q,\{3\}}(?X, ?Y) \rightarrow \text{answer}_{Q,\{3\}}(?X, ?Y) \]
From SPARQL to TriQ

\[ Q = (?X, \text{name, } ?Y) \text{ OPT } (?X, \text{phone, } ?Z) \]

\[ M_Q = [ (\emptyset, t_{\text{bgp}}(Q)), (t_{\text{opr}}(Q), t_{\text{out}}(Q)) ] \]

Given an RDF graph \( G \):

\[ \text{evaluation of } Q \text{ over } G = \text{ans}(M_Q, DB(G)) \]
From SPARQL to TriQ

\[ Q = (?X, \text{name, ?Y}) \text{ OPT (?X, phone, ?Z)} \]

\[ M_Q = [ (\emptyset, \tau_{\text{bgp}}(Q)), (\tau_{\text{opr}}(Q), \tau_{\text{out}}(Q)) ] \]

Given an RDF graph \( G \):

\[
\text{evaluation of } Q \text{ over } G = \text{ans}(M_Q, DB(G))
\]

\[
\{ \text{triple}(a, b, c) | (a, b, c) \text{ belongs to } G \}
\]
From SPARQL over OWL 2 QL to TriQ

\[ G = \{(\text{dog, rdf:type, animal}), (\text{animal, rdfs:subClassOf, } \exists \text{eats})\} \]

- Elements of \( G \) that eat something: SELECT \(?X\) (?X, eats, ?Y)

- However, the answer is empty due to the active domain semantics
From SPARQL over OWL 2 QL to TriQ

\[ G = \{(\text{dog, rdf:type, animal}), (\text{animal, rdfs:subClassOf, } \exists\text{eats})\} \]

• Elements of \( G \) that eat something:  \textbf{SELECT} ?X \ (\text{?X, eats, ?Y})

• However, the answer is empty due to the \textbf{active domain semantics}

• We need the query \((\text{?X, rdf:type, } \exists\text{eats})\):

\[ G \models_{\text{OWL2}} (\text{dog, rdf:type, } \exists\text{eats}) \]

• This is what is called “the evaluation of \( Q \) over \( G \) under the OWL 2 direct semantics entailment regime”
From SPARQL over OWL 2 QL to TriQ

\[ G = \{(\text{dog, rdf:type, animal}), (\text{animal, rdfs:subClassOf, } \exists \text{eats})\} \]

\[ Q = (?X, \text{rdf:type, } \exists \text{eats}) \]

\[ M_Q = [ (\tau_{\text{OWL2QL}}, \tau_{\text{bgp}}(Q)), (\emptyset, \tau_{\text{out}}(Q)) ] \]

fixed program used to encode the semantics \( \models_{\text{OWL2}} \)

query\(_Q(\text{?X}) \rightarrow \text{answer}_Q(\text{?X}) \)

\[ \text{triple}_1(\text{?X, rdf:type, } \exists \text{eats}), \text{dom}(\text{?X}) \rightarrow \text{query}_Q(\text{?X}) \]
From SPARQL over OWL 2 QL to TriQ

**Theorem:** Given a SPARQL query $Q$ and an RDF graph $G$:

The evaluation of $Q$ over $G$ under the OWL 2 direct semantics entailment regime $= \text{ans}(M_Q, DB(G))$

- $M_Q = [(\tau_{\text{OWL2QL}}, \tau_{\text{bgp}}(Q)), (\tau_{\text{opr}}(Q), \tau_{\text{out}}(Q))]$ is a TriQ query
- $\tau_{\text{OWL2QL}}$ is fixed, it does not depend on $Q$
Active Domain Semantics Revisited

\[ G = \{(\text{dog, rdf:type, animal}), (\text{animal, rdfs:subClassOf, } \exists \text{eats})\} \]

- Elements of $G$ that eat something: SELECT ?X (?X, eats, ?Y)
- However, the answer is empty due to the active domain semantics
- We need the query (?X, rdf:type, $\exists$eats)
Active Domain Semantics Revisited

\[ G = \{(\text{dog}, \text{rdf:type}, \text{animal}), (\text{animal}, \text{rdfs:subClassOf}, \exists \text{eats})\} \]

\[ Q = (?X, \text{rdf:type, } \exists \text{eats}) \]

\[ M_Q = [ (T_{\text{OWL2QL}}, T_{\text{bgp}}(Q)), (\emptyset, T_{\text{out}}(Q)) ] \]

\( triple_1(?X, \text{rdf:type}, \exists \text{eats}), \text{dom}(?X) \rightarrow \text{query}_Q(?X) \)
Active Domain Semantics Revisited

\[ G = \{(\text{dog, rdf:type, animal}), (\text{animal, rdfs:subClassOf, } \exists\text{eats})\} \]

\[ Q = \text{SELECT } ?X \quad (?X, \text{eats, } ?Y) \]

\[ M_Q = [ (T_{\text{OWL2QL}}, T_{\text{bgp}}(Q)), (T_{\text{opr}}(Q), T_{\text{out}}(Q)) ] \]

\[ \text{triple}_1(?X, \text{eats, } ?Y), \quad \text{dom}(?X), \text{dom}(?Y) \rightarrow \text{query}_P(?X, ?Y) \]

\[ \text{query}_P(?X, ?Y) \rightarrow \text{query}_Q(?X) \]
Active Domain Semantics Revisited

\[ G = \{(\text{dog}, \text{rdf:type}, \text{animal}), (\text{animal}, \text{rdfs:subClassOf}, \exists \text{eats})\} \]

\[ Q = \text{SELECT } ?X (\{ ?X \text{, eats, } ?Y \} \bigcup ?X, ?Y) \]

\[ M_Q = [ (\tau_{\text{OWL2QL}}, \tau_{\text{bgp}}(Q)), (\tau_{\text{opr}}(Q), \tau_{\text{out}}(Q)) ] \]

\[ \text{triple}_1(?X, \text{eats, } ?Y), \quad \text{dom}(?X), \text{dom}(?Y) \rightarrow \text{query}_P(?X, ?Y) \]

\[ \text{query}_P(?X, ?Y) \rightarrow \text{query}_Q(?X) \]
Active Domain Semantics Revisited

\[ G = \{(\text{dog, rdf:type, animal}), (\text{animal, rdfs:subClassOf, } \exists \text{eats})\} \]

\[ Q = \text{SELECT } ?X \ (?X, \text{eats, } ?Y) \]

\[ M_Q = \left[ (T_{\text{OWL2QL}}, T_{\text{bgp}}(Q)), (\emptyset, T_{\text{out}}(Q)) \right] \]

\[ \text{triple}_1(?X, \text{eats, } ?Y), \text{dom}(?X) \rightarrow \text{query}_P(?X) \]
Active Domain Semantics Revisited

\[ G = \{(\text{dog}, \text{rdf:type, animal}), (\text{animal}, \text{rdfs:subClassOf, } \exists \text{eats})\} \]

\[ Q = \text{SELECT } ?X \ ( ?X, \text{eats, } ?Y) \]

\[ Q = ( ?X, \text{eats, } _:b) \]

\[ M_Q = [ (T_{\text{OWL2QL}}, T_{\text{bgp}}(Q)), (\emptyset, T_{\text{out}}(Q)) ] \]

\[ triple_1(?X, \text{eats, } ?Y), \text{dom}(?X) \rightarrow query_{\rho}(?X) \]
The Language TriQ-Lite

join variables occur only at non-affected positions
The Language TriQ-Lite

**Theorem:** Every SPARQL query under the entailment regime for OWL 2 QL can be expressed as a TriQ-Lite query (with or without the active domain restriction).
The Language TriQ-Lite

**Theorem:** Query evaluation for TriQ-Lite is PTIME-complete

**Proof sketch:**

**PTIME-membership:** enough to consider the evaluation problem for weakly-guarded constant-join Datalog[$\exists, \neg s, \bot$]

- Let $D$ be an RDF graph and $(\Pi, \Lambda)$ be a weakly-guarded constant-join Datalog[$\exists, \neg s, \bot$] query

- First step: Constraints are eliminated from $\Pi$ to generate $\text{ex}(\Pi)$
  - Checking for inconsistencies

- Second step: ground chase is computed
  - Atoms $p(a_1, \ldots, a_k)$ in chase($D$, $\text{ex}(\Pi)$) such that $a_1, \ldots, a_k$ are URIs
  - ALOGSPACE algorithm
The Language TriQ-Lite

**Theorem:** Query evaluation for TriQ-Lite is PTIME-complete

**Proof sketch:**

- Third step: negation is eliminated from ex(\(\Pi\)) to generate \(D_L\) and \(\text{ex}(\Pi)^+\)
  - Every atom \(\neg p(t_1, \ldots, t_k)\) in a rule of \(\text{ex}(\Pi)\) is replaced by \(\text{cp}(t_1, \ldots, t_k)\), where \(\text{cp}\) stores the complement of \(p\)
  - \(D_L\) is the extension of \(D\) with the atoms \(\text{cp}(a_1, \ldots, a_k)\) such that \(a_1, \ldots, a_k\) are URIs, which are computed in each strata of \(\text{ex}(\Pi)\) by using the ground chase

- Last step: transform \(\text{ex}(\Pi)^+\) into a linear Datalog[∃] program \(\Pi_L\)
  - Key observation: in every rule of \(\text{ex}(\Pi)^+\), every variable that can be assigned non-URI values must occur only in the weak-guard.
The Language TriQ-Lite

**Theorem:** Query evaluation for TriQ-Lite is PTIME-complete

**Proof sketch:**

- To finish the proof: query evaluation problem for linear Datalog\[\exists\] is in PTIME in program complexity
  - Program complexity: only $\Lambda$ is fixed, program $\Pi_L$ and RDF graph $D_L$ depend on $\Pi$ and $D$

**PTIME-hardness:** since Datalog is already PTIME-hard
Concluding remarks

1. We introduce the modular query language TriQ
   • TripQ captures EXPTIME (no order)

2. We show that every SPARQL query can be expressed in TriQ
   • Including the OWL 2 direct semantics entailment regime
   • Dropping the active domain restriction

3. We identify the tractable fragment TriQ-Lite of TriQ with the same properties as in 2.

4. We also prove that the existential quantification in TriQ-Lite is necessary
   • In the paper, we define and study some notions of program expressiveness
Thank you!
Backup slides
RDFS and OWL Vocabularies

Can be even worse…

(dbpedia:Ullman, owl:sameAs, yago:Ullman)
(yago:Ullman, name, “Jeffrey Ullman”)

::

SELECT ?X

((?Y, rdf:type, ?Z) AND
(?Z, rdf:type, owl:Restriction) AND
(?Z, owl:onProperty, is_author_of) AND
(?Z, owl:someValuesFrom, owl:Thing)

AND (?Y, name, ?X))

UNION

((?Y, is_author_of, ?Z) AND (?Y, owl:sameAs, ?W)
AND (?W, name, ?X)))
The Program $\tau_{\text{OWL2QL}}$

- Collect the domain elements:
  
  $$\text{triple}(?X, ?Y, ?Z) \rightarrow \text{dom}(?X), \text{dom}(?Y), \text{dom}(?Z)$$

- Store the elements in the ontology:
  
  $$\text{triple}(?X, \text{rdf:type}, ?Y) \rightarrow \text{type}(?X, ?Y)$$
  
  $$\text{triple}(?X, \text{rdfs:subPropertyOf}, ?Y) \rightarrow \text{sp}(?X, ?Y)$$
  
  $$\text{triple}(?X, \text{owl:inverseOf}, ?Y) \rightarrow \text{inv}(?Y, ?X)$$
  
  $$\text{triple}(?X, \text{owl:Restriction}, ?Y) \rightarrow \text{rest}(?X, ?Y)$$
  
  $$\text{triple}(?X, \text{rdfs:subClassOf}, ?Y) \rightarrow \text{sc}(?X, ?Y)$$
  
  $$\text{triple}(?X, \text{owl:DisjointWith}, ?Y) \rightarrow \text{disj}(?X, ?Y)$$
  
  $$\text{triple}(?X, ?Y, ?Z) \rightarrow \text{triple}_1(?X, ?Y, ?Z)$$
The Program $\tau_{\text{OWL2QL}}$

- **Reason about properties:**

  \[
  \text{sp}(\text{?X}, \text{?Y}), \text{inv}(\text{?Z}, \text{?X}), \text{inv}(\text{?W}, \text{?Y}) \rightarrow \text{sp}(\text{?Z}, \text{?W})
  \]

  \[
  \text{type}(\text{?X}, \text{owl:ObjectProperty}) \rightarrow \text{sp}(\text{?X}, \text{?X})
  \]

  \[
  \text{sp}(\text{?X}, \text{?Y}), \text{sp}(\text{?Y}, \text{?Z}) \rightarrow \text{sp}(\text{?X}, \text{?Z})
  \]

- **Reason about classes:**

  \[
  \text{sp}(\text{?X}, \text{?Y}), \text{rest}(\text{?Z}, \text{?X}), \text{rest}(\text{?W}, \text{?Y}) \rightarrow \text{sc}(\text{?Z}, \text{?W})
  \]

  \[
  \text{type}(\text{?X}, \text{owl:Class}) \rightarrow \text{sc}(\text{?X}, \text{?X})
  \]

  \[
  \text{sc}(\text{?X}, \text{?Y}), \text{sc}(\text{?Y}, \text{?Z}) \rightarrow \text{sc}(\text{?X}, \text{?Z})
  \]

- **Reason about disjointness constraints:**

  \[
  \text{disj}(\text{?X}, \text{?Y}), \text{sc}(\text{?Z}, \text{?X}), \text{sc}(\text{?W}, \text{?Y}) \rightarrow \text{disj}(\text{?Z}, \text{?W})
  \]
The Program $\tau_{\text{OWL2QL}}$

- Reason about membership assertions:


\[\text{type}(?X, ?Y), \text{rest}(?Y, ?U) \rightarrow \exists ?Z \text{ triple}_1(?X, ?U, ?Z)\]

\[\text{type}(?X, ?Y), \text{sc}(?Y, ?Z) \rightarrow \text{type}(?X, ?Z)\]

\[\text{type}(?X, ?Y) \rightarrow \text{triple}_1(?X, \text{rdf:type}, ?Y)\]

\[\text{triple}_1(?X, ?U, ?Y), \text{rest}(?Z, ?U) \rightarrow \text{type}(?X, ?Z)\]

\[\text{type}(?X, ?Y), \text{type}(?X, ?Z), \text{disj}(?Y, ?Z) \rightarrow \bot\]
Active Domain Semantics Revisited

\[ G = \{(\text{dog, rdf:type, animal}), (\text{animal, rdfs:subClassOf, } \exists \text{eats})\} \]

Dropping the active domain semantics in SPARQL is non-trivial:

Consider the query \( (?X, \text{rdfs:subClassOf, } ?Y) \)
Is TriQ-Lite really necessary?

• Is existential quantification really necessary?

• TriQ-Lite is Datalog rewritable - it seems that it is not

• But, what about our main objective - need for decoupling?
Is TriQ-Lite really necessary?

\[
M_Q = [(\tau_{\text{OWL2QL}}, t_{\text{bgp}}(Q)), (\tau_{\text{opr}}(Q), t_{\text{out}}(Q))] \\
M_{Q,\Pi} = [(\Pi, t_{\text{bgp}}(Q)), (\tau_{\text{opr}}(Q), t_{\text{out}}(Q))] \\
\]

**Theorem:** There exists an RDF graph \( G \) and a SPARQL query \( Q \) such that, for every Datalog[\(\neg s, \bot\)] program \( \Pi \):

\[
\text{ans}(M_{Q,\Pi}, DB(G)) \neq \text{ans}(M_Q, DB(G))
\]
Is TriQ-Lite really necessary?

Proof sketch:

RDF graph G:

(c, rdfs:subClassOf, ∃p)

(∃p^-, rdfs:subClassOf, c)

(a, rdf:type, c)
Is TriQ-Lite really necessary?

Proof sketch:

RDF graph G:

\( (c, \text{rdfs:subClassOf}, \exists p) \)

\( \exists p^-, \text{rdfs:subClassOf}, c \)  

\( (a, \text{rdf:type}, c) \)

\( c(?X) \rightarrow \exists ?Y \ p(?X, ?Y) \)

\( p(?X, ?Y) \rightarrow c(?Y) \)

\( c(a) \)
Is TriQ-Lite really necessary?

Proof sketch:

RDF graph G:

\[(c, \text{rdfs:subClassOf}, \exists p)\]

\[(\exists p^-, \text{rdfs:subClassOf}, c)\]

\[(a, \text{rdf:type}, c)\]

SPARQL query Q:

```
```
Is TriQ-Lite really necessary?

Proof sketch:

RDF graph G:

(c, rdfs:subClassOf, ∃p)
(∃p⁻, rdfs:subClassOf, c)
(a, rdf:type, c)

SPARQL query Q:

Is TriQ-Lite really necessary?

Proof sketch:

RDF graph G:

(c, rdfs:subClassOf, ∃p)  
(∃p⁻, rdfs:subClassOf, c)

(a, rdf:type, c)

SPARQL query Q:

UNION

Is TriQ-Lite really necessary?

Proof sketch:

RDF graph G:

(c, rdfs:subClassOf, ∃p)  Answer: a
(∃p⁻, rdfs:subClassOf, c)  In a Datalog[¬s, ⊥] program: a is an
(a, rdf:type, c)          answer if and only if there exists a
URI b such that (a,b) is an answer

SPARQL query Q:

Program Expressive Power

**Pep**: captures the expressive power of a program
Program Expressive Power

**Pep**: captures the expressive power of a program

\[
\text{Pep}_\Omega[\Pi] = \{ (D, \Lambda, p(a_1, \ldots, a_k)) \mid (\Pi, \Lambda) \text{ is a program in } \Omega \text{ and } p(a_1, \ldots, a_k) \text{ is in ans}((\Pi, \Lambda), D) \}
\]

\[
\text{Pep}[\Omega] = \{ \text{Pep}_\Omega[\Pi] \mid \Pi \text{ is a program in } \Omega \}
\]

\(\Omega_1\) is more expressive than \(\Omega_2\) if \(\text{Pep}[\Omega_2]\) is a proper subset of \(\text{Pep}[\Omega_1]\)
Program Expressive Power

**Pep**: captures the expressive power of a program

\[
Pep_\Omega[\Pi] = \{ (D, \Lambda, p(a_1, ..., a_k)) \mid (\Pi, \Lambda) \text{ is a program in } \Omega \text{ and } p(a_1, ..., a_k) \text{ is in ans}((\Pi, \Lambda), D) \} \]

\[
Pep[\Omega] = \{ \text{Pep}_\Omega[\Pi] \mid \Pi \text{ is a program in } \Omega \} \]

\[\Omega_1 \text{ is more expressive than } \Omega_2 \text{ if } \text{Pep}[\Omega_2] \text{ is a proper subset of } \text{Pep}[\Omega_1] \]

**Theorem**: weakly-guarded constant-join Datalog\[[\exists, \neg s, \bot] \] is more expressive than Datalog\[[\neg s, \bot] \]