Expressive Languages for Querying the Semantic Web

Marcelo Arenas\textsuperscript{1}  Georg Gottlob\textsuperscript{2}  Andreas Pieris\textsuperscript{2}

\textsuperscript{1}Department of Computer Science, PUC Chile, Chile
\textsuperscript{2}Department Computer Science, University of Oxford, UK

PODS 2014, Snowbird, Utah, USA
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)

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

```
dbpedia:Ullman is_author_of “Database Systems: The Complete Book"
```

```
“Jeffrey Ullman” name
```

```
dbpedia:Aho is_coauthor_of
```

```
“Alfred Aho” name
```

```
dbpedia:Ullman name
```

```
dbpedia:Aho
```

```
“Jeffrey Ullman”
```
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)*

  (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

```sparql
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, Arenas & 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)  \[ r₁ = \{ a \mid \text{there exists a URI } b \text{ such that } (a, \text{is_coauthor_of}, b) \} \]
(r₁, owl:onProperty, is_coauthor_of)
(r₁, owl:someValuesFrom, owl:Thing)

(r₂, rdf:type, owl:Restriction)  \[ r₂ = \{ a \mid \text{there exists a URI } b \text{ such that } (a, \text{is_author_of}, b) \} \]
(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”)

(r_1, rdf:type, owl:Restriction)
(r_1, owl:onProperty, is_coauthor_of)
(r_1, owl:someValuesFrom, owl:Thing)

(r_2, rdf:type, owl:Restriction)
(r_2, owl:onProperty, is_author_of)
(r_2, owl:someValuesFrom, owl:Thing)

(r_1, rdfs:subClassOf, r_2)

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”)

(r_1, rdf:type, owl:Restriction)
(r_1, owl:onProperty, is_coauthor_of)
(r_1, owl:someValuesFrom, owl:Thing)

(r_2, rdf:type, owl:Restriction)
(r_2, owl:onProperty, is_author_of)
(r_2, owl:someValuesFrom, owl:Thing)

(r_1, rdfs:subClassOf, r_2)

SELECT ?X

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

Expected answers: Jeffrey Ullman
Alfred Aho
RDFS and OWL Vocabularies

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

Expected answers: Jeffrey Ullman ✓
Alfred Aho ✗

(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)

SELECT ?X

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

Expected answers: Jeffrey Ullman ✓
Alfred Aho ✗

we are forced to encode the semantics
of RDFS and OWL in the query
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”)

\[
\begin{align*}
(r_1, \text{rdf:type}, \text{owl:Restriction}) \\
(r_1, \text{owl:onProperty}, \text{is_coauthor_of}) \\
(r_1, \text{owl:someValuesFrom}, \text{owl:Thing}) \\
(r_2, \text{rdf:type}, \text{owl:Restriction}) \\
(r_2, \text{owl:onProperty}, \text{is_author_of}) \\
(r_2, \text{owl:someValuesFrom}, \text{owl:Thing}) \\
(r_1, \text{rdfs:subClassOf}, r_2)
\end{align*}
\]

\[
\text{SELECT ?X}
\]

\[
\begin{align*}
(?Y, \text{rdf:type}, ?Z) \ \text{AND} \\
(?Z, \text{rdf:type}, \text{owl:Restriction}) \ \text{AND} \\
(?Z, \text{owl:onProperty}, \text{is_author_of}) \ \text{AND} \\
(?Z, \text{owl:someValuesFrom}, \text{owl:Thing}) \\
\text{AND} \ (?Y, \text{name}, \ ?X)
\end{align*}
\]

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
Our Objectives

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

![Diagram]

- **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

• Is TriQ-Lite really necessary?

• Concluding remarks
Triple Query Language (TriQ)

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

\[ ans(M, G) = ans(Q_{SPARQL}, ans(Q_{RDFS/OWL}, G)) \]
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 \( Q_{\text{SPARQL}} \)?
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\([\neg s]\) represents every SPARQL query
- Datalog\([\exists, \perp]\) is appropriate for ontological reasoning
Triple Query Language (TriQ)

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

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

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

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

**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


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

Affected positions $= ?$
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 = \(\{T[3], P[1], Q[2]\}\)
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

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

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

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

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

\[
\]

\[
\]

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

Affected positions = \{T[3], P[1], Q[2], T[2], P[2]\}
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

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

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

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

Affected positions = \{T[3], P[1], Q[2], T[2], P[2], 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

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

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

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

Affected positions = \{T[3], P[1], Q[2], T[2], P[2], 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

\[ P(?X, ?Y), S(\text{X} ?Z) \rightarrow \exists?W \ T(?Y, ?X, ?W) \]
\[ T(?X, ?Y, ?Z) \rightarrow \exists?W \ P(?W, ?Y) \]
\[ P(?X, ?Y), \neg R(?X) \rightarrow \exists?Z \ Q(?X, ?Z) \]

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

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

Affected positions = \{T[3], P[1], Q[2], T[2], P[2], 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)

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

weakly-guarded Datalog[∃, ¬s, ⊥] 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: \(Φ(x_1, \ldots, x_k) \rightarrow answer(x_1, \ldots, x_k)\)
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[$\exists, \neg$] (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)
From SPARQL to TriQ

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

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 \( a \)
From SPARQL to TriQ

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

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 \(a\).

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

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

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

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

\[ \text{triple}(\?X, \text{name}, \?Y) \rightarrow \text{query}_A(\?X, \?Y) \]

\[ \text{triple}(\?X, \text{phone}, \?Y) \rightarrow \text{query}_B(\?X, \?Y) \]
From SPARQL to TriQ

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

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

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

list of individuals with phone number

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

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

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

\[
\begin{align*}
\text{query}_A(?X, ?Y), \text{query}_B(?X, ?Z) & \rightarrow \text{query}_Q(?X, ?Y, ?Z) \\
\text{query}_A(?X, ?Y), \text{query}_B(?X, ?Z) & \rightarrow \text{compatible}_Q(?X) \\
\text{query}_A(?X, ?Y), \neg \text{compatible}_Q(?X) & \rightarrow \text{query}_{Q,(3)}(?X, ?Y)
\end{align*}
\]

list of individuals with phone number

the third argument (phone number) is missing
From SPARQL to TriQ

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

A

B

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

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

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

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

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

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

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

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

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

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

• However, the answer is empty due to the \text{active domain semantics}
From SPARQL over OWL 2 QL to TriQ

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

- Elements of \(G\) that eat something: \text{SELECT} \ ?X \ (?X, \text{eats}, \ ?Y)\
- However, the answer is empty due to the active domain semantics\
- We need the query (\(?X, \text{rdf:type}, \exists \text{eats}\)):
  \[ G \models_{\text{OWL2}} (\text{dog}, \text{rdf:type}, \exists \text{eats})\]
- 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

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

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

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

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

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

\( \text{triple}_1(\text{?X, rdf:type, } \exists \text{eats}), \text{dom}(\text{?X}) \rightarrow \text{query}_Q(\text{?X}) \)
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 $= ans(M_Q, DB(G))$

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

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

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

- However, the answer is empty due to the \text{active domain semantics}

- We need the query (\text{?X, rdf:type, \exists eats})
Active Domain Semantics Revisited

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

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

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

\[ \text{triple}_1(\?X, \text{rdf:type, } \exists \text{eats}), \text{dom}(\?X) \rightarrow \text{query}_Q(\?X) \]
Active Domain Semantics Revisited

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

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

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

\( \text{query}_P(?X, ?Y) \rightarrow \text{query}_Q(?X) \)

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

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

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

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

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

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

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

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

\[ M_Q = [ (\text{r} \text{OWL2QL}, \text{Tbgp}(Q)), (\emptyset, \text{Tout}(Q)) ] \]

\[ triple_1(?X, \text{eats, } ?Y), \text{dom}(?X) \rightarrow query_p(?X) \]
Active Domain Semantics Revisited

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

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

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

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

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

constant-join programs

PTIME-complete

EXPTIME-complete

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 \( \text{ex}(\Pi) \) 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 \( cp(t_1, \ldots, t_k) \), where \( cp \) stores the complement of \( p \)
  - \( D_L \) is the extension of \( D \) with the atoms \( cp(a_1, \ldots, a_k) \) 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[∃] 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
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?

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

\[ M_{Q,\Pi} = \left[ (\Pi, \tau_{\text{bgp}}(Q)), (\tau_{\text{opr}}(Q), \tau_{\text{out}}(Q)) \right] \]

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

\[ \text{ans}(M_{Q,\Pi}, DB(G)) \neq \text{ans}(M_Q, DB(G)) \]
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 prove that the existential quantification in TriQ-Lite is necessary
   • In the paper, we define and study some other notions of program expressiveness
Thank you!
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:

  $\text{triple(?X, ?Y, ?Z)} \rightarrow \text{dom(?X), dom(?Y), dom(?Z)}$

- Store the elements in the ontology:

  $\text{triple(?X, rdf:type, ?Y)} \rightarrow \text{type(?X, ?Y)}$

  $\text{triple(?X, rdfs:subPropertyOf, ?Y)} \rightarrow \text{sp(?X, ?Y)}$

  $\text{triple(?X, owl:inverseOf, ?Y)} \rightarrow \text{inv(?Y, ?X)}$

  $\text{triple(?X, owl:Restriction, ?Y)} \rightarrow \text{rest(?X, ?Y)}$

  $\text{triple(?X, rdfs:subClassOf, ?Y)} \rightarrow \text{sc(?X, ?Y)}$

  $\text{triple(?X, owl:DisjointWith, ?Y)} \rightarrow \text{disj(?X, ?Y)}$

  $\text{triple(?X, ?Y, ?Z)} \rightarrow \text{triple$_1$(?X, ?Y, ?Z)}$
The Program $\tau_{\text{OWL2QL}}$

- Reason about properties:


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

  $sp(?X, ?Y), sp(?Y, ?Z) \rightarrow sp(?X, ?Z)$

- Reason about classes:


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

  $sc(?X, ?Y), sc(?Y, ?Z) \rightarrow sc(?X, ?Z)$

- Reason about disjointness constraints:

The Program $\tau_{\text{OWL2QL}}$

- Reason about membership assertions:

\[
\begin{align*}
\text{type}(?X, ?Y), \; \text{rest}(?Y, ?U) & \rightarrow \exists ?Z \; \text{triple}_1(?X, ?U, ?Z) \\
\text{type}(?X, ?Y), \; \text{sc}(?Y, ?Z) & \rightarrow \text{type}(?X, ?Z) \\
\text{type}(?X, ?Y) & \rightarrow \text{triple}_1(?X, \text{rdf:type}, ?Y) \\
\text{triple}_1(?X, ?U, ?Y), \; \text{rest}(?Z, ?U) & \rightarrow \text{type}(?X, ?Z) \\
\text{type}(?X, ?Y), \; \text{type}(?X, ?Z), \; \text{disj}(?Y, ?Z) & \rightarrow \bot
\end{align*}
\]
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, rdfs:subClassOf, ?Y)
Is TriQ-Lite really necessary?

Proof sketch:

RDF graph G:

(c, rdfs:subClassOf, \exists p)

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

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

Proof sketch:

RDF graph G:

(c, rdfs:subClassOf, ∃p)       c(?X) → ∃?Y p(?X, ?Y)
(∃p−, rdfs:subClassOf, c)       p(?X, ?Y) → c(?Y)
(a, rdf:type, c)               c(a)
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, $\exists p$)

(\$\exists p^\neg\$, 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)
(a, rdf:type, c)

SPARQL query Q:

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

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

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

\(\Omega_1\) 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

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

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

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

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