Expressive Languages for Querying the Semantic Web

Marcelo Arenas[1] Georg Gottlob[2] Andreas Pieris[2]

[1]Department of Computer Science, PUC Chile, Chile

[2]Department Computer Science, University of Oxford, UK

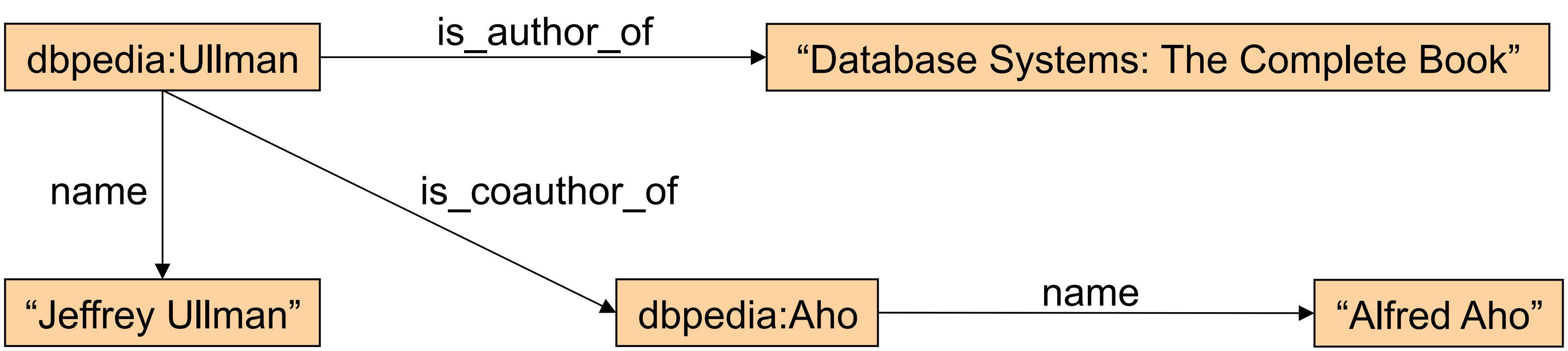
Dagstuhl 2014

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*)



dbpedia: <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 - (*subject*, *predicate*, *object*)

(dbpedia:Ullman, is_author_of, “Database Systems: The Complete Book”)

(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]

SPARQL

SELECT ?X

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

(dbpedia:Ullman, is_author_of, “Database Systems: The Complete Book”)

(dbpedia:Ullman, name, “Jeffrey Ullman”)

(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)

(dbpedia:Aho, name, “Alfred Aho”)

SPARQL

SELECT ?X

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

(dbpedia:Ullman, is_author_of, “Database Systems: The Complete Book”)

(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, is_author_of, “Database Systems: The Complete Book”)

(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)

RDFS and OWL Vocabularies

(dbpedia:Ullman, is_author_of, “Database Systems: The Complete Book”)

(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_1 = {*a* | there exists a URI *b* such that (*a*, is_coauthor_of, *b*)}

(r_2, rdf:type, owl:Restriction)

(r_2, owl:onProperty, is_author_of)

(r_2, owl:someValuesFrom, owl:Thing)

r_2 = {*a* | there exists a URI *b* such that (*a*, is_author_of, *b*)}

(r_1, rdfs:subClassOf, r_2)

RDFS and OWL Vocabularies

(dbpedia:Ullman, is_author_of, “Database Systems: The Complete Book”)

(dbpedia:Ullman, name, “Jeffrey Ullman”)

(dbpedia:Aho, is_coauthor_of, dbpedia:Ullman)

(dbpedia:Aho, name, “Alfred Aho”)

$$\left.\begin{array}{l}(r_1,\ \text{rdf:type},\ \text{owl:Restriction})\\(r_1,\ \text{owl:onProperty},\ \text{is\_coauthor\_of})\\(r_1,\ \text{owl:someValuesFrom},\ \text{owl:Thing})\\ \\(r_2,\ \text{rdf:type},\ \text{owl:Restriction})\\(r_2,\ \text{owl:onProperty},\ \text{is\_author\_of})\\(r_2,\ \text{owl:someValuesFrom},\ \text{owl:Thing})\\ \\(r_1,\ \text{rdfs:subClassOf},\ r_2)\end{array}\right\}\quad \text{each co-author is an author}$$

RDFS and OWL Vocabularies

(dbpedia:Ullman, is_author_of, “Database Systems: The Complete Book”)

(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

(dbpedia:Ullman, is_author_of, “Database Systems: The Complete Book”)

(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

(dbpedia:Ullman, is_author_of, “Database Systems: The Complete Book”)

(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, 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, is_author_of, “Database Systems: The Complete Book”)

(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, 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)

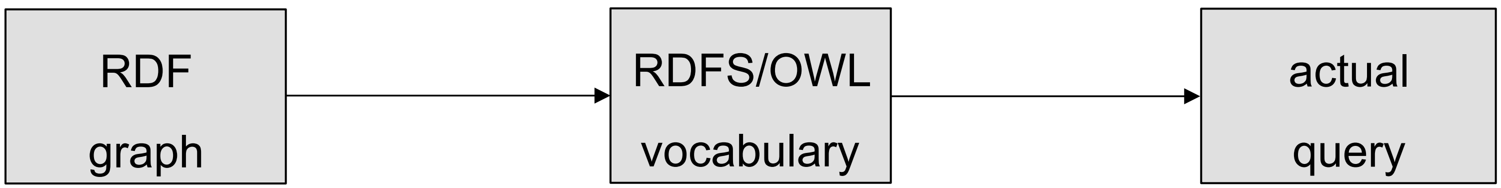
AND (?Y, name, ?X))

Our Objectives

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

Our Objectives

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



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)

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

$$ans(M,G) = ans(Q_{\mathrm{SPARQL}},\ ans(Q_{\mathrm{RDFS/OWL}},\ G))$$

Triple Query Language (TriQ)

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

$$ans(M,G) = ans(Q_{\mathrm{SPARQL}},\ ans(Q_{\mathrm{RDFS/OWL}},\ G))$$

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

Triple Query Language (TriQ)

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

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

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

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

Datalog[∃, ¬s, ⊥]

Triple Query Language (TriQ)

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

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

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

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

$\left.\begin{array}{l} \text{Datalog}[\neg s] \\ \text{Datalog}[\exists, \bot] \end{array}\right\}$ Datalog[∃, ¬s, ⊥]

weakly-guarded Datalog[∃, ¬s, ⊥]

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 = ?

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]}

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]}

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]

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[∃,¬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[∃,¬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

$P(?X, ?Y), S(\cancel{?Y}, ?Z) \rightarrow \exists ?W\ T(?Y, ?X, ?W)$

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

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

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

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

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

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

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

weakly-guarded Datalog[∃, ¬s] + $\Phi(x_1, \ldots, x_k) \rightarrow \perp$

=

weakly-guarded Datalog[∃, ¬s, ⊥]

Triple Query Language (TriQ)

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

weakly-guarded Datalog[$\exists$, $\neg$s, $\bot$] queries

Triple Query Language (TriQ)

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

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: $\Phi(x_1, \ldots, x_k) \rightarrow \mathit{answer}(x_1, \ldots, x_k)$

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[∃,¬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)

From SPARQL to TriQ

$$Q \;=\; \underbrace{(?X,\ \text{name},\ ?Y)}_{A}\ \ \text{OPT}\ \underbrace{(?X,\ \text{phone},\ ?Z)}_{B}$$

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

$$Q \;=\; \underbrace{(?X, \text{name}, ?Y)}_{A} \;\; \text{OPT} \; \underbrace{(?X, \text{phone}, ?Z)}_{B}$$

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*

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

evaluate basic graph patterns (*A* and *B*)

output rules

encode the semantics of SPARQL operators (OPT)

From SPARQL to TriQ

$$Q \;=\; \underbrace{(?X,\ \text{name},\ ?Y)}_{A}\ \ \text{OPT}\ \underbrace{(?X,\ \text{phone},\ ?Z)}_{B}$$

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

$$\mathit{triple}(?X,\ \text{name},\ ?Y)\ \rightarrow\ \mathit{query}_A(?X,\ ?Y)$$

$$\mathit{triple}(?X,\ \text{phone},\ ?Y)\ \rightarrow\ \mathit{query}_B(?X,\ ?Y)$$

From SPARQL to TriQ

$$Q \;=\; \underbrace{(?X,\ \mathrm{name},\ ?Y)}_{A}\ \ \mathrm{OPT}\ \underbrace{(?X,\ \mathrm{phone},\ ?Z)}_{B}$$

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

$$query_A(?X, ?Y),\ query_B(?X, ?Z) \;\rightarrow\; query_Q(?X, ?Y, ?Z)$$

list of individuals with phone number

$$query_A(?X, ?Y),\ query_B(?X, ?Z) \;\rightarrow\; compatible_Q(?X)$$

From SPARQL to TriQ

$$Q \;=\; \underbrace{(?X,\ \text{name},\ ?Y)}_{A}\ \ \text{OPT}\ \underbrace{(?X,\ \text{phone},\ ?Z)}_{B}$$

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

$$query_A(?X, ?Y),\ query_B(?X, ?Z) \ \rightarrow\ query_Q(?X, ?Y, ?Z)$$

list of individuals with phone number

$$query_A(?X, ?Y),\ query_B(?X, ?Z) \ \rightarrow\ compatible_Q(?X)$$

the third argument (phone number) is missing

$$query_A(?X, ?Y),\ \neg compatible_Q(?X) \ \rightarrow\ query_{Q,\{3\}}(?X, ?Y)$$

From SPARQL to TriQ

$$Q \;=\; \underbrace{(?X,\ \text{name},\ ?Y)}_{A}\ \ \text{OPT}\ \underbrace{(?X,\ \text{phone},\ ?Z)}_{B}$$

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

$$query_{Q}(?X, ?Y, ?Z) \rightarrow \; answer_{Q}(?X, ?Y, ?Z)$$

$$query_{Q,\{3\}}(?X, ?Y) \;\rightarrow\; answer_{Q,\{3\}}(?X, ?Y)$$

From SPARQL to TriQ

$$Q \;=\; \underbrace{(?X,\ \text{name},\ ?Y)}_{A}\ \ \text{OPT}\ \underbrace{(?X,\ \text{phone},\ ?Z)}_{B}$$

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

$$query_{Q,\emptyset}(?X, ?Y, ?Z) \rightarrow answer_{Q,\emptyset}(?X, ?Y, ?Z)$$

$$query_{Q,\{3\}}(?X, ?Y) \rightarrow answer_{Q,\{3\}}(?X, ?Y)$$

From SPARQL to TriQ

$$Q \;=\; \underbrace{(?X,\ \text{name},\ ?Y)}_{A}\ \ \text{OPT}\ \underbrace{(?X,\ \text{phone},\ ?Z)}_{B}$$

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

Given an RDF graph G:

evaluation of Q over G $=$ $ans(M_Q,\ DB(G))$

From SPARQL to TriQ

$$Q \;=\; \underbrace{(?X,\ \text{name},\ ?Y)}_{A}\ \ \text{OPT}\ \underbrace{(?X,\ \text{phone},\ ?Z)}_{B}$$

$$M_Q \;=\; [\ (\varnothing,\ \tau_{bgp}(Q)),\ (\tau_{opr}(Q),\ \tau_{out}(Q))\ ]$$

Given an RDF graph G:

evaluation of Q over G = $ans(M_Q, DB(G))$

$\{\ triple(a, b, c) \mid (a, b, c)$ belongs to $G\}$

From SPARQL over OWL 2 QL to TriQ

G = {(dog, rdf:type, animal), (animal, rdfs:subClassOf, ∃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

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

- 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, ∃eats):

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

- 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 = {(dog, rdf:type, animal), (animal, rdfs:subClassOf, ∃eats)}

Q = (?X, rdf:type, ∃eats)

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

fixed program used to encode the semantics $\models_{OWL2}$

$query_Q$(?X) → $answer_Q$(?X)

$triple_1$(?X, rdf:type, ∃eats), dom(?X) → $query_Q$(?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 $= \mathit{ans}(M_Q, \mathit{DB}(G))$

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

Active Domain Semantics Revisited

G = {(dog, rdf:type, animal), (animal, rdfs:subClassOf, ∃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, ∃eats)

Active Domain Semantics Revisited

G = {(dog, rdf:type, animal), (animal, rdfs:subClassOf, ∃eats)}

Q = (?X, rdf:type, ∃eats)

$$M_Q = [\ (\tau_{OWL2QL}, \tau_{bgp}(Q)), (\varnothing, \tau_{out}(Q))\ ]$$

$triple_1$(?X, rdf:type, ∃eats), dom(?X) → $query_Q$(?X)

Active Domain Semantics Revisited

G = {(dog, rdf:type, animal), (animal, rdfs:subClassOf, ∃eats)}

$$Q \;=\; \text{SELECT ?X } \underbrace{\text{(?X, eats, ?Y)}}_{P}$$

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

triple$_1$(?X, eats, ?Y),
dom(?X), *dom*(?Y) → *query*$_P$(?X,?Y)

query$_P$(?X,?Y) → *query*$_Q$(?X)

Active Domain Semantics Revisited

G = {(dog, rdf:type, animal), (animal, rdfs:subClassOf, ∃eats)}

$$Q = \text{SELECT ?X } \underbrace{(\text{?X, eats, ?Y})}_{P}$$

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

$triple_1(?X, \text{eats}, ?Y),$
$dom(?X), \cancel{dom(?Y)} \rightarrow query_P(?X,?Y)$

$query_P(?X,?Y) \rightarrow query_Q(?X)$

Active Domain Semantics Revisited

G = {(dog, rdf:type, animal), (animal, rdfs:subClassOf, ∃eats)}

Q = SELECT ?X (?X, eats, ?Y)

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

$$\downarrow$$

$$triple_1(?X, eats, ?Y), dom(?X) \rightarrow query_P(?X)$$

Active Domain Semantics Revisited

G = {(dog, rdf:type, animal), (animal, rdfs:subClassOf, ∃eats)}

Q = SELECT ?X (?X, eats, ?Y)

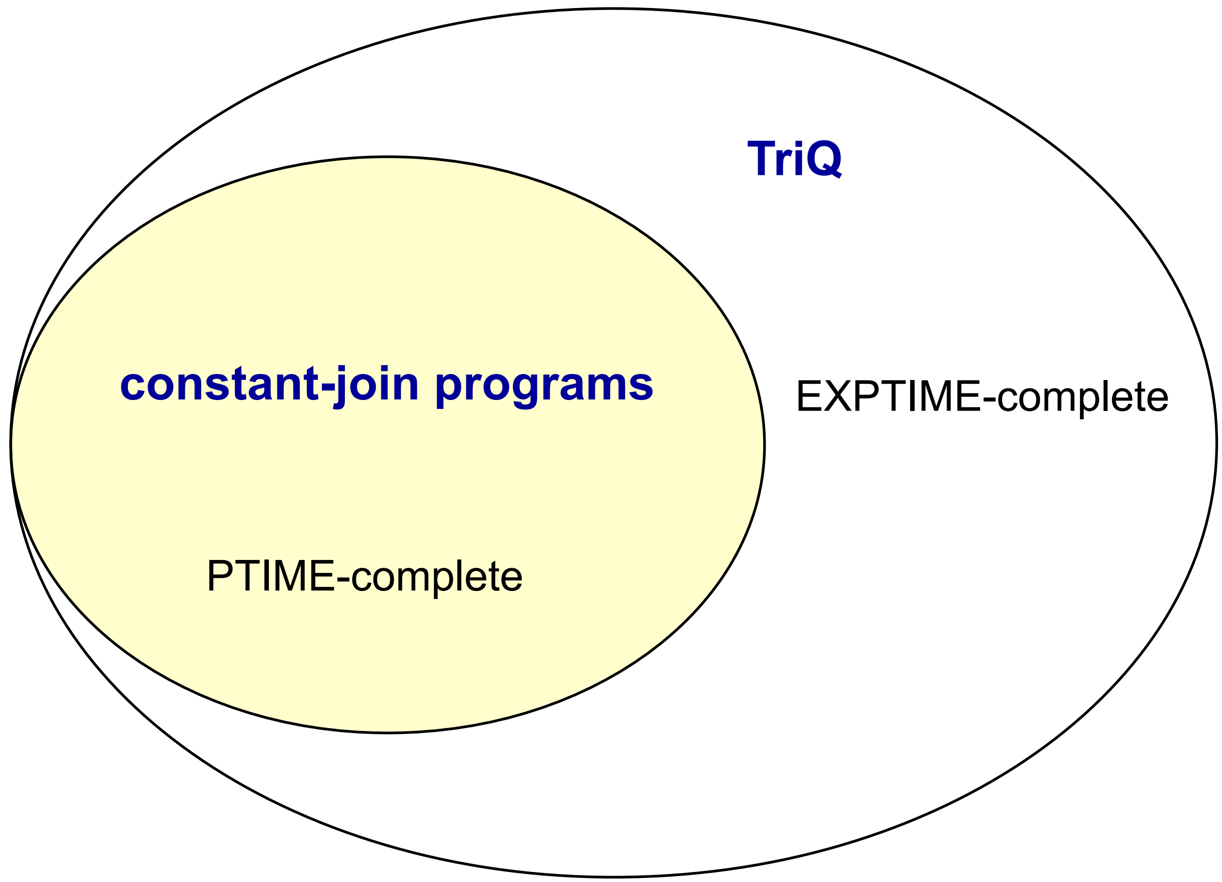
Q = (?X, eats, _:b)

$$M_{Q} = [\ (\tau_{OWL2QL}, \tau_{bgp}(Q)), (\emptyset, \tau_{out}(Q))\ ]$$

$\downarrow$

$triple_1$(?X, eats, ?Y), dom(?X) → $query_P$(?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[∃, ¬s, ⊥]

  - Let D be an RDF graph and (Π, Λ) be a weakly-guarded constant-join Datalog[∃, ¬s, ⊥] query

- First step: Constraints are eliminated from Π to generate ex(Π)
  - Checking for inconsistencies

- Second step: ground chase is computed
  - Atoms $p(a_1, \ldots, a_k)$ in chase(D, ex(Π)) 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(Π) to generate D_L and $ex(\Pi)^+$
  - Every atom $\neg p(t_1, \ldots, t_k)$ in a rule of ex(Π) 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 ex(Π) by using the ground chase

- Last step: transform $ex(\Pi)^+$ into a linear Datalog[∃] program Π_L
  - Key observation: in every rule of $ex(\Pi)^+$, 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 Λ is fixed, program Π_L and RDF graph D_L depend on Π 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, is_author_of, “Database Systems: The Complete Book”)

(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 τ_{OWL2QL}

- Collect the domain elements:

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

- Store the elements in the ontology:

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

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

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

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

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

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

  $$triple(?X, ?Y, ?Z) \rightarrow triple_1(?X, ?Y, ?Z)$$

The Program τ_{OWL2QL}

- Reason about properties:

  sp(?X, ?Y), *inv*(?Z, ?X), *inv*(?W, ?Y) → *sp*(?Z, ?W)

  type(?X, owl:ObjectProperty) → *sp*(?X, ?X)

  sp(?X, ?Y), *sp*(?Y, ?Z) → *sp*(?X, ?Z)

- Reason about classes:

  sp(?X, ?Y), *rest*(?Z, ?X), *rest*(?W, ?Y) → *sc*(?Z, ?W)

  type(?X, owl:Class) → *sc*(?X, ?X)

  sc(?X, ?Y), *sc*(?Y, ?Z) → *sc*(?X, ?Z)

- Reason about disjointness constraints:

  disj(?X, ?Y), *sc*(?Z, ?X), *sc*(?W, ?Y) → *disj*(?Z, ?W)

The Program τ_{OWL2QL}

- Reason about membership assertions:

$$triple_1(?X, ?U, ?Y),\ sp(?U, ?V) \rightarrow triple_1(?X, ?V, ?Y)$$

$$triple_1(?X, ?U, ?Y),\ inv(?U, ?V) \rightarrow triple_1(?Y, ?V, ?X)$$

$$type(?X, ?Y),\ rest(?Y, ?U) \rightarrow \exists ?Z\ triple_1(?X, ?U, ?Z)$$

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

$$type(?X, ?Y) \rightarrow triple_1(?X, \text{rdf:type}, ?Y)$$

$$triple_1(?X, ?U, ?Y),\ rest(?Z, ?U) \rightarrow type(?X, ?Z)$$

$$type(?X, ?Y),\ type(?X, ?Z),\ disj(?Y, ?Z) \rightarrow \perp$$

Active Domain Semantics Revisited

G = {(dog, rdf:type, animal), (animal, rdfs:subClassOf, ∃eats)}

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

Consider the query (?X, 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?

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

$$M_{Q,\Pi} \;=\; [\, (\Pi, \tau_{bgp}(Q)), (\tau_{opr}(Q), \tau_{out}(Q)) \,]$$

Theorem: There exists an RDF graph *G* and a SPARQL query *Q* such that, for every Datalog[¬s, ⊥] program Π:

$$ans(M_{Q,\Pi}, DB(G)) \;\neq\; ans(M_Q, DB(G))$$

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

Is TriQ-Lite really necessary?

Proof sketch:

RDF graph G**:**

(c, rdfs:subClassOf, $\exists$p)

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

(a, rdf:type, c)

SPARQL query Q**:**

SELECT ?X ((?X, p, ?U) AND (?U, p, ?V)) UNION

SELECT ?X ?Y ((?X, p, ?W) AND (?W, p, ?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)

SPARQL query Q**:**

SELECT ?X ((?X, p, ?U) AND (?U, p, ?V))   UNION

SELECT ?X ?Y ((?X, p, ?W) AND (?W, p, ?Y))

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**:**

SELECT ?X ((?X, p, ?U) AND (?U, p, ?V)) UNION

SELECT ?X ?Y ((?X, p, ?W) AND (?W, p, ?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)

Answer: a

In a Datalog[$\neg$s, $\perp$] program: a is an answer if and only if there exists a URI b such that (a,b) is an answer

SPARQL query Q**:**

SELECT ?X ((?X, p, ?U) AND (?U, p, ?V)) UNION

SELECT ?X ?Y ((?X, p, ?W) AND (?W, p, ?Y))

Program Expressive Power

Pep: captures the expressive power of a program

Program Expressive Power

Pep: captures the expressive power of a program

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

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

Ω_1 is more expressive than Ω_2 if $\mathrm{Pep}[\Omega_2]$ is a proper subset of $\mathrm{Pep}[\Omega_1]$

Program Expressive Power

Pep: captures the expressive power of a program

$\mathrm{Pep}_{\Omega}[\Pi]$ = { $(D, \Lambda, p(a_1, \ldots, a_k))$ | (Π, Λ) is a program in Ω and $p(a_1, \ldots, a_k)$ is in $\mathrm{ans}((\Pi, \Lambda), D)$ }

$\mathrm{Pep}[\Omega]$ = { $\mathrm{Pep}_{\Omega}[\Pi]$ | Π is a program in Ω }

Ω_1 is more expressive than Ω_2 if $\mathrm{Pep}[\Omega_2]$ is a proper subset of $\mathrm{Pep}[\Omega_1]$

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