Solving SPARQL Query Containment

using (Sub-) Graph Isomorphisms

René Schubotz [1]

[1] German Research Center for Artificial Intelligence

April 25, 2019

Overview

Resource Description Framework

Let $\mathbf{I}$, $\mathbf{L}$ and $\mathbf{B}$ be pairwise disjoint infinite sets of `IRI`s, literals and blank nodes.

The infinite set of all *RDF triples* is $\mathcal{T} = \mathbf{IB} \times \mathbf{I} \times \mathbf{IBL}$.

An *RDF graph* $G \subset \mathcal{T}$ is a finite set of RDF triples.

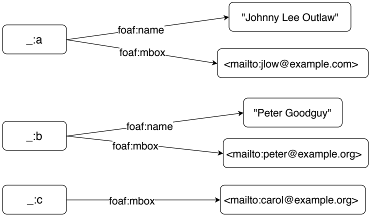
An *RDF dataset* $\mathcal{D}$ is a set $\{G_0, \langle u_1, G_1\rangle, \ldots, \langle u_n, G_n\rangle\}$, where G_0 is the default graph, and $G_1, \ldots, G_n$ are named graphs with names $u_1, \ldots, u_n \in \mathbf{I}$.

Resource Description Framework

Example: RDF graph

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a   foaf:name    "Johnny Lee Outlaw" .
_:a   foaf:mbox    <mailto:jlow@example.com> .
_:b   foaf:name    "Peter Goodguy" .
_:b   foaf:mbox    <mailto:peter@example.org> .
_:c   foaf:mbox    <mailto:carol@example.org> .
```



A **simple interpretation** $\mathcal{I}$ is a tuple $\mathcal{I} = (\mathbb{R}, \mathbb{P}, \circ_{\mathsf{Int}}, \circ_{\mathsf{Ext}}, \circ_{\mathsf{L}})$ such that

- $\mathbb{R}$ is a non-empty set of resources, called the domain or also universe of discourse of $\mathcal{I}$,
- $\mathbb{P}$ is the set of all properties[1],
- $\circ_{\mathbf{Ext}} : \mathbb{P} \to 2^{\mathbb{R}\times\mathbb{R}}$ is a mapping that assigns a binary relational extension to each property,
- $\circ_{\mathbf{Int}} : \mathbf{I} \to (\mathbb{R} \cup \mathbb{P})$ is the interpretation mapping that assigns a resource or property to a given `IRI`,
- $\circ_{\mathbf{L}} : \mathbf{L} \to \mathbb{R}$ is a partial mapping from literals into the set of literal values[2]

[1] Not necessarily disjoint from or a subset of $\mathbb{R}$.

[2] The set of literal values is a subset of the set of resources $\mathbb{R}$.

A simple interpretation $\mathcal{I}$ is a *model* of an RDF graph G iff

- If $e \in \mathbf{L}$, then $\mathcal{I}(e) = \circ_{\mathbf{L}}(e)$.
- If $e \in \mathbf{IB}$, then $\mathcal{I}(e) = \circ_{\mathbf{Int}}(e)$.
- If $e = (s, p, o) \in \mathbf{IB} \times \mathbf{I} \times \mathbf{IBL}$, then $\mathcal{I}(e) = \top$ iff $\mathcal{I}(p) \in \mathbb{P}$ and $(\mathcal{I}(s), \mathcal{I}(o)) \in \circ_{\mathbf{Ext}}(\mathcal{I}(p))$.
- If e is an RDF graph, then $\mathcal{I}(e) = \bot$ iff $\mathcal{I}(e') = \bot$ for some RDF triple $e' \in e$, otherwise $\mathcal{I}(e) = \top$.

An RDF graph G is (simply) *satisfiable* if a simple interpretation $\mathcal{I}$ exists with $\mathcal{I}(G) = \top$, otherwise G is (simply) *unsatisfiable*.

We say that the RDF graph G (simply) *entails* the RDF graph H ($G \models H$) if and only if every model $\mathcal{I}$ which satisfies G also satisfies H.

Two RDF graphs G and H are *equivalent* ($G \equiv H$) if and only if $G \models H$ and $H \models G$.

An RDF graph G is considered *lean* if and only if there does not exist a proper subgraph $G' \subset G$ such that $G' \models G$.

More RDF Entailment Regimes

More RDF Entailment Regimes

Literals and Datatypes

An *RDF datatype* d is defined as a 4-tuple $(\mathcal{L}_d, \mathcal{V}_d, \mu_d, \mathbf{D}_d)$, where $\mathcal{L}_d$ is its lexical space, $\mathcal{V}_d$ is its value space, μ_d gives the datatype's lexical-to-value mapping, and $\mathbf{D}_d$ is the set of its datatype IRIs.

The subset $\mathbf{L}$ of RDF terms is called *RDF literals* and characterized as $\mathbf{L} = \mathbf{L}_t \cup \mathbf{L}_l$ where $\mathcal{L}$ is the infinite set of **lexical forms** and $\mathbf{L}_t = (\mathcal{L} \times \binom{\mathbf{D}}{1})$ with $\mathbf{D}$ is the set of RDF datatype IRIs, and $\mathbf{L}_l = (\mathcal{L} \times \{\mathtt{rdf:langString}\} \times \mathcal{T})$ with $\mathcal{T}$ is the is the set of non-empty and well-formed **language tags**.

More RDF Entailment Regimes

D-Interpretation, D-Satisfiability and D-Entailment

Let $\mathbf{D}$ be the set of datatype IRIs. A *(simple) D-interpretation* $\mathcal{I}_{\mathbf{D}}$ is a s-interpretation satisfying

- $\circ_{\mathbf{L}}(l) = \mu_{\mathcal{I}(\mathrm{ddd})}(\mathrm{sss}) = \mu_d(\mathrm{sss}) \iff l = (\mathrm{sss}, \mathrm{ddd}) \in \mathbf{L}_t$,
- $\circ_{\mathbf{L}}(l) = (\mathrm{sss}, \mathrm{ttt}) \iff l = (\mathrm{sss}, \mathtt{rdf:langString}, \mathrm{ttt}) \in \mathbf{L}_l$,
- $\circ_{\mathbf{L}}(l) = \bot$, otherwise

under the condition that $\mathrm{sss} \in \mathcal{L}$, $\mathrm{ttt} \in \mathcal{T}$, $\mathtt{rdf:langString} \in \mathbf{D}$ and $\mathrm{ddd} \in \mathbf{D}$.

An RDF graph G is (simply) *D-satisfiable* if a D-interpretation $\mathcal{I}_{\mathbf{D}}$ exists with $\mathcal{I}_{\mathbf{D}}(G) = \top$, otherwise G is (simply) *D-unsatisfiable*.

An RDF graph G (simply) *D-entails* an RDF graph H, when every D-interpretation which D-satisfies G also D-satisfies H.

More RDF Entailment Regimes

RDF Vocabulary and Axiomatic Triples

The *RDF Vocabulary* $\mathsf{Voc}_{\mathsf{RDF}}$ is an infinite set of RDF terms given as follows:

$$\mathsf{Voc}_{\mathsf{RDF}} = \{ \quad \texttt{rdf:type}, \texttt{rdf:subject}, \texttt{rdf:predicate}, \texttt{rdf:object}, \texttt{rdf:first}, \texttt{rdf:rest}, \texttt{rdf:value}, \texttt{rdf:nil}, \texttt{rdf:List}, \texttt{rdf:langString}, \texttt{rdf:Property}, \texttt{rdf:\_1}, \texttt{rdf:\_2}, \ldots \}$$

The infinite set of *axiomatic RDF triples* is given as follows:

$$\{ \quad (\texttt{rdf:type}, \texttt{rdf:type}, \texttt{rdf:Property}), (\texttt{rdf:subject}, \texttt{rdf:type}, \texttt{rdf:Property}), (\texttt{rdf:predicate}, \texttt{rdf:type}, \texttt{rdf:Property}), (\texttt{rdf:object}, \texttt{rdf:type}, \texttt{rdf:Property}), (\texttt{rdf:first}, \texttt{rdf:type}, \texttt{rdf:Property}), (\texttt{rdf:rest}, \texttt{rdf:type}, \texttt{rdf:Property}), (\texttt{rdf:value}, \texttt{rdf:type}, \texttt{rdf:Property}), (\texttt{rdf:nil}, \texttt{rdf:type}, \texttt{rdf:List}), (\texttt{rdf:\_1}, \texttt{rdf:type}, \texttt{rdf:Property}), (\texttt{rdf:\_2}, \texttt{rdf:type}, \texttt{rdf:Property}), \ldots \}$$

More RDF Entailment Regimes

RDF-Interpretation, RDF-Satisfiability and RDF-Entailment

An *RDF-interpretation recognising D* is a (simple) D-interpretation $\mathcal{I}_{\mathbf{D}}$ with $\texttt{xsd:string} \in \mathbf{D}$ and $\texttt{rdf:langString} \in \mathbf{D}$ satisfying the following conditions

- $(x, \mathcal{I}_{\mathbf{D}}(\texttt{rdf:Property})) \in \circ_{\mathbf{Ext}}(\mathcal{I}_{\mathbf{D}}(\texttt{rdf:type})) \iff x \in \mathbb{P}$
- $\forall \mathrm{ddd} \in \mathbf{D} : (x, \mathcal{I}_{\mathbf{D}}(\mathrm{ddd})) \in \circ_{\mathbf{Ext}}(\mathcal{I}_{\mathbf{D}}(\texttt{rdf:type})) \iff x \in \mathcal{V}_{\mathcal{I}_{\mathbf{D}}(\mathrm{ddd})}$

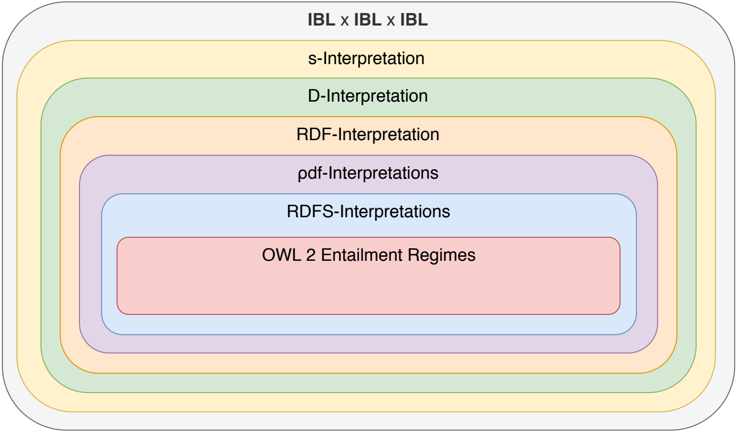
as well as every RDF triple in the infinite set of *RDF axioms*.

An RDF graph G is *RDF-satisfiable* (recognising $\mathbf{D}$) if there exists an RDF-interpretation (recognising $\mathbf{D}$) with $\mathcal{I}_{\mathbf{D}}(G) = \top$, otherwise G is *RDF-unsatisfiable*.

An RDF graph G *RDF-entails* an RDF graph H (recognising $\mathbf{D}$), when every RDF-interpretation (recognising $\mathbf{D}$) which RDF-satisfies G also RDF-satisfies H.

More RDF Entailment Regimes

Overview



Graph Morphisms and RDF

Graph Morphisms

Overview

$$\begin{array}{ccc} \text{Homomorphism} & \Leftarrow & \text{Isomorphism} \\ \Uparrow & & \Uparrow \\ \text{Endomorphism} & \Leftarrow & \text{Automorphism} \end{array}$$

Graph Homomorphism

$$\begin{array}{ccc} \textbf{Homomorphism} & \Leftarrow & \text{Isomorphism} \\ \Uparrow & & \Uparrow \\ \text{Endomorphism} & \Leftarrow & \text{Automorphism} \end{array}$$

Given two undirected graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, a mapping $\beta : V_G \to V_H$ is a *homomorphism* from G to H if and only if

$$(u, v) \in E_G \implies (\beta(u), \beta(v)) \in E_H$$

Graph Isomorphism

Definition

$$\begin{array}{ccc} \text{Homomorphism} & \Leftarrow & \textbf{Isomorphism} \\ \Uparrow & & \Uparrow \\ \text{Endomorphism} & \Leftarrow & \text{Automorphism} \end{array}$$

Given two undirected graphs $G = (V_G, E_G)$ and $H = (V_H, E_H)$, a bijection $\beta : V_G \to V_H$ is an *isomorphism* from G to H if and only if

$$(u, v) \in E_G \Leftrightarrow (\beta(u), \beta(v)) \in E_H$$

Graph Endomorphism

Definition

$$\begin{array}{ccc} \text{Homomorphism} & \Leftarrow & \text{Isomorphism} \\ \Uparrow & & \Uparrow \\ \textbf{Endomorphism} & \Leftarrow & \text{Automorphism} \end{array}$$

Given an undirected graph $G = (V_G, E_G)$, a mapping $\beta : V_G \to V_G$ is an *endomorphism* if and only if it is an homomorphism from G to G.

Graph Automorphism

Definition

$$
\begin{array}{ccc}
\text{Homomorphism} & \Leftarrow & \text{Isomorphism} \\
\Uparrow & & \Uparrow \\
\text{Endomorphism} & \Leftarrow & \textbf{Automorphism}
\end{array}
$$

Given an undirected graph $G = (V_G, E_G)$, a mapping $\beta : V_G \to V_G$ is an *automorphism* if and only if it is an isomorphism from G to G.

Morphisms on RDF Graphs

Blank node mappings and RDF homomorphisms

$$\begin{array}{ccc} \textbf{Homomorphism} & \Leftarrow & \text{Isomorphism} \\ \Uparrow & & \Uparrow \\ \text{Endomorphism} & \Leftarrow & \text{Automorphism} \end{array}$$

Let $\mu : \mathbf{ILB} \to \mathbf{ILB}$ be a partial mapping of RDF terms to RDF terms. If μ is the identity on **IL**, we call it a *blank node mapping*.

Two RDF graphs G and H are *homomorphic* if and only if there exists a blank node mapping μ such that $\mu(H) \subseteq G$.

$$\begin{array}{ccc} \text{Homomorphism} & \Leftarrow & \textbf{Isomorphism} \\ \Uparrow & & \Uparrow \\ \text{Endomorphism} & \Leftarrow & \text{Automorphism} \end{array}$$

Two RDF graphs G and H are *isomorphic*, denoted $G \cong H$, if and only if there exists a *bijective* blank node mapping μ such that $\mu(G) = H$, in which case we call μ an *isomorphism*.

Morphisms on RDF Graphs

RDF automorphism

$$\begin{array}{ccc} \text{Homomorphism} & \Leftarrow & \text{Isomorphism} \\ \Uparrow & & \Uparrow \\ \text{Endomorphism} & \Leftarrow & \textbf{Automorphism} \end{array}$$

A blank node mapping μ is an *automorphism* of an RDF graph G if $\mu(G) = G$. If μ is the identity mapping on blank nodes in G then μ is a *trivial automorphism*; otherwise μ is a *non-trivial automorphism*. We denote the set of all automorphisms of G by $Aut(G)$.

Morphisms on RDF Graphs

RDF endomorphism

$$\begin{array}{ccc} \text{Homomorphism} & \Leftarrow & \text{Isomorphism} \\ \Uparrow & & \Uparrow \\ \textbf{Endomorphism} & \Leftarrow & \text{Automorphism} \end{array}$$

Given an RDF graph G, we denote by $End(G)$ all blank node mappings with domain $terms(G)$ that map G to a subgraph of itself:

$$End(G) = \{\mu \mid \mu(G) \subseteq G \wedge dom(\mu) = terms(G)\}$$

If $\mu(G) \subset G$, we call it a proper endomorphism.

Morphisms on RDF Graphs

RDF core endomorphism

$$
\begin{array}{ccc}
\text{Homomorphism} & \Leftarrow & \text{Isomorphism} \\
\Uparrow & & \Uparrow \\
\textbf{Endomorphism} & \Leftarrow & \text{Automorphism}
\end{array}
$$

We denote by $\mathit{CEnd}(G) \subset \mathit{End}(G)$ the set of all mappings that map to the fewest unique blank nodes

$$\mathit{CEnd}(G) = \{\mu \in \mathit{End}(G) \mid \nexists \mu' \in \mathit{End}(G) : |\mathit{codom}(\mu') \cap \mathbf{B}| < |\mathit{codom}(\mu) \cap \mathbf{B}|\}$$

We call $\mathit{CEnd}(G)$ the *core endomorphisms* of G.

RDF Morphisms and s-Entailment

We say that the RDF graph G *s-entails* the RDF graph H ($G \models H$) if and only if every model $\mathcal{I}$ which satisfies G also satisfies H.

An RDF graph G *s-entails* an RDF graph H ($G \models H$) if and only if $\exists \mu \in Hom(H) : \mu(H) \subseteq G$.

An RDF graph G is *s-equivalent* with an RDF graph H if it holds that G and H are isomorphic, i.e. $(G \cong H) \Rightarrow (G \equiv H)$. However, $(G \equiv H) \nRightarrow (G \cong H)$!

An RDF graph G is *s-equivalent* with an RDF graph H if and only if $\exists \mu \in CEnd(G), \mu' \in CEnd(H) : \mu(G) \cong \mu'(H)$.

Morphisms, s-Entailment and Query Answering

Given a query graph $q \in \mathbf{IB} \times \mathbf{IB} \times \mathbf{IBL}$ and an RDF graph $G \subset \mathbf{IB} \times \mathbf{I} \times \mathbf{IBL}$.

We say that G *answers* q if and only if $G \models q$.

The set $\{\mu : \mathbf{IBL} \rightarrow \mathbf{ILB} \mid \mu = id_{\mathbf{IL}} \ \wedge \ \exists G' \subseteq G : \mu(q) \cong G'\}$ is the set of all solution mappings of q with respect to G.

SPARQL Protocol and RDF Query Language

SPARQL Protocol and RDF Query Language

SPARQL graph pattern expressions: Syntax

Let $\mathbf{V}$ denote the infinite set of variables that is disjoint from $\mathbf{IBL}$. *SPARQL graph pattern* expressions are defined recursively as:

(1) A triple pattern $t \in \mathbf{IBV} \times \mathbf{IBV} \times \mathbf{IBLV}$ is a graph pattern.

(2) If P_1 and P_2 are graph patterns, then expression $(P_1 \texttt{ AND } P_2)$, $(P_1 \texttt{ UNION } P_2)$, $(P_1 \texttt{ OPT } P_2)$ and $(P_1 \texttt{ MINUS } P_2)$ are graph patterns.

(3) If P is a graph pattern and R is a SPARQL built-in condition, then the expression $(P \texttt{ FILTER } R)$ is a graph pattern.

(4) If P is a graph pattern and $a \in \mathbf{IV}$, then the expression $(a \texttt{ GRAPH } P)$ is a graph pattern.

(5) If P is a graph pattern and $a \in \mathbf{IV}$, then the expression $(a \texttt{ SERVICE } P)$ is a graph pattern.

The semantics of SPARQL graph patterns is defined in terms of an evaluation function $[\![\cdot]\!]_G^{\mathcal{D}}$.

$[\![\cdot]\!]_G^{\mathcal{D}}$ returns a *set of mappings* for a given SPARQL graph pattern expression, a fixed and *active dataset* $\mathcal{D}$ and an *active graph* G within $\mathcal{D}$.

A *mapping* is a partial function $\mu : \mathbf{V} \to \mathbf{IBL}$.

Two mappings μ_1 and μ_2 are *compatible* (written as $\mu_1 \sim \mu_2$) if $\forall v \in dom(\mu_1) \cap dom(\mu_2) : \mu_1(v) = \mu_2(v)$.

The mapping with empty domain, denoted by $\mu_\emptyset$, is compatible with every other mapping. If $\mu_1 \sim \mu_2$, then we write $\mu_1 \cup \mu_2$ for

the mapping obtained by *extending* μ_1 according to μ_2 on all the variables in $dom(\mu_2) \setminus dom(\mu_1)$.

Given two sets of mappings Ω_1 and Ω_2, we define

(1) $(\Omega_1 \bowtie \Omega_2) = \{\mu_1 \cup \mu_2 \mid \mu_1 \in \Omega_1,\ \mu_2 \in \Omega_2,\ \mu_1 \sim \mu_2\}$

(2) $(\Omega_1 \cup \Omega_2) = \{\mu \mid \mu \in (\Omega_1 \cup \Omega_2)\}$

(3) $(\Omega_1 \setminus \Omega_2) = \{\mu \in \Omega_1 \mid \forall \mu' \in \Omega_2 : \mu \not\sim \mu'\}$

(4) $(\Omega_1 ⟕ \Omega_2) = (\Omega_1 \bowtie \Omega_2) \cup (\Omega_1 \setminus \Omega_2)$

(5) $\pi_V(\Omega) = \{\mu' \mid \exists \mu \in \Omega : \mu' \subseteq \mu, dom(\mu') = V \cap dom(\mu)\}$

SPARQL Protocol and RDF Query Language I

SPARQL graph pattern expressions: Semantics

Evaluation $[\![P]\!]_G^{\mathcal{D}}$ of a SPARQL graph pattern P over a dataset $\mathcal{D}$ with active graph G is defined recursively as

(1) If P is a triple pattern, then $[\![P]\!]_G^{\mathcal{D}} = \{\mu : var(P) \to \mathbf{IBL} \mid \mu(P) \in G\}$.

(2) If $P = (P_1 \texttt{ AND } P_2)$, then $[\![P]\!]_G^{\mathcal{D}} = [\![P_1]\!]_G \bowtie [\![P_2]\!]_G$.

(3) If $P = (P_1 \texttt{ UNION } P_2)$, then $[\![P]\!]_G^{\mathcal{D}} = [\![P_1]\!]_G \cup [\![P_2]\!]_G$.

(4) If $P = (P_1 \texttt{ OPT } P_2)$, then $[\![P]\!]_G^{\mathcal{D}} = [\![P_1]\!]_G ⟕ [\![P_2]\!]_G$.

(5) If $P = (P_1 \texttt{ MINUS } P_2)$, then $[\![P]\!]_G^{\mathcal{D}} = [\![P_1]\!]_G \setminus [\![P_2]\!]_G$.

(6) If $P = (P_1 \texttt{ FILTER } R)$, then $[\![P]\!]_G = \{\mu \in [\![P_1]\!]_G \mid \mu \models R\}$.

(7) If $P = (g \; \texttt{GRAPH} \; P_1)$ with $g \in \mathbf{IV}$, then

$$[\![P]\!]_G^{\mathcal{D}} = \begin{cases} [\![P_1]\!]_{\sigma_{\mathcal{D}}[g]}^{\mathcal{D}} \Leftrightarrow g \in \nu_{\mathcal{D}} \\ \bigcup_{u \in \mathbf{I}} ([\![P_1]\!]_{\sigma_{\mathcal{D}}[u]}^{\mathcal{D}} \bowtie \{[g \mapsto u]\}) \Leftrightarrow g \in \mathbf{V} \end{cases}$$

(8) If $P = (s \; \texttt{SERVICE} \; P_1)$ with $s \in \mathbf{IV}$, then

$$[\![P]\!]_G^{\mathcal{D}} = \begin{cases} [\![P_1]\!]_{\sigma_{\mathcal{D}}[s]}^{\mathcal{D}} \Leftrightarrow s \in \mathbf{I} \\ \{\mu_\varnothing\} \Leftrightarrow s \notin \nu_{\mathcal{D}} \\ TBD \end{cases}$$

A SELECT query $\mathbf{q}$ is an expression of the form

$$\mathbf{q} = \mathtt{SELECT}_V\ \mathtt{WHERE}\ \{\ P\ \}$$

where $V \subseteq \mathit{var}(P)$, and P is a SPARQL graph pattern.

The semantics of a SELECT query over an input dataset $\mathcal{D}$ with active graph G is given as

$$\mathit{ans}(\mathbf{q}, \mathcal{D}) = \pi_V(\llbracket P \rrbracket_G^{\mathcal{D}})$$

RDF Graph

```
@prefix foaf:  <http://xmlns.com/foaf/0.1/> .

_:a   foaf:name   "Johnny Lee Outlaw" .
_:a   foaf:mbox   <mailto:jlow@example.com> .
_:b   foaf:name   "Peter Goodguy" .
_:b   foaf:mbox   <mailto:peter@example.org> .
_:c   foaf:mbox   <mailto:carol@example.org> .
```

SPARQL Query

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?name ?mbox
WHERE { ?x foaf:name ?name ; foaf:mbox ?mbox . }
```

Result

name	mbox
"Johnny Lee Outlaw"	mailto:jlow@example.com
"Peter Goodguy"	mailto:peter@example.org

A CONSTRUCT query $\mathbf{q}$ is an expression of the form

$$\mathbf{q} = \texttt{CONSTRUCT}\ H\ \texttt{WHERE}\ \{\ P\ \}$$

where $H \subset \mathbf{IBV} \times \mathbf{IBV} \times \mathbf{IBLV}$ is a finite set of triple patterns, called a template, and P is a SPARQL graph pattern.

The semantics of a CONSTRUCT query over an input dataset $\mathcal{D}$ with active graph G is given as

$$ans(\mathbf{q}, \mathcal{D}) = \{\mu(t) \| \mu \in [\![P]\!]_G^{\mathcal{D}}, t \in H\}$$

SPARQL Protocol and RDF Query Language

Example

RDF Graph

```
@prefix org:    <http://example.com/ns#> .

_:a  org:employeeName   "Alice" .
_:a  org:employeeId     12345 .
_:b  org:employeeName   "Bob" .
_:b  org:employeeId     67890 .
```

SPARQL Query

```
PREFIX foaf:   <http://xmlns.com/foaf/0.1/>
PREFIX org:    <http://example.com/ns#>

CONSTRUCT { ?x foaf:name ?name }
WHERE   { ?x org:employeeName ?name }
```

Result

```
@prefix org: <http://example.com/ns#> .

_:x foaf:name "Alice" .
_:y foaf:name "Bob" .
```

SPARQL Query Containment and Equivalence

SPARQL Query Containment

Definition

Query containment is the problem of deciding whether the result set of one query is included in the result set of another.

Given two queries q_1 and q_2, we say that q_2 *contains* q_1 iff

$$q_1 \sqsubseteq q_2 \Leftrightarrow \forall G \subset \mathcal{T} : ans(\mathbf{q_1}, \{G\}) \subseteq ans(\mathbf{q_2}, \{G\})$$

Note that this definition does *not* depend on a specific RDF graph or RDF dataset!

SPARQL Query Containment

Example

SPARQL Query q_1

```
PREFIX ex: <http://example.com/ns#>

SELECT ?x ?y ?z
WHERE {
  ?x ex:sister ?y .
  ?y ex:name ?z .
}
```

SPARQL Query q_2

```
SELECT *
WHERE { ?s ?p ?o. }
```

SPARQL Query Equivalence

Definition

Query equivalence is the problem of deciding whether the result set of one query is equivalent to the result set of another.

Given two queries q_1 and q_2, we say that q_1 and q_2 are *equivalent* iff

$$q_1 \equiv q_2 \Leftrightarrow q_1 \sqsubseteq q_2 \wedge q_2 \sqsubseteq q_1$$

Note that definition of query containment and equivalence does *not* depend on a specific RDF graph or RDF dataset!

SPARQL Query Equivalence

Example

SPARQL Query q_1

```
PREFIX ex: <http://example.com/ns#>

SELECT ?z
WHERE {
   { ?x ex:sister ?y .
     ?y ex:name ?z . }
   { ?w ex:mother ?x . }
   UNION
   { ?w ex:father ?x. }
}
```

SPARQL Query q_2

```
PREFIX ex: <http://example.com/ns#>

SELECT ?n
WHERE {
   { ?s :name ?n .
     ?c :mother ?p .
     ?p :sister ?s . }
   UNION
   { ?s :name ?n .
     ?c :father ?p .
     ?p :sister ?s . }
}
```

SPARQL Query Containment & Equivalence

Motivation

Rewriting: find a *minimal* query equivalent to the original one

Optimization: reduce the computational cost of query evaluation using minimized queries

Caching: increase throughput of query processing by re-using work done for (containing) previous queries

Composition: perform (partial) composition of `CONSTRUCT` queries q_1 and q_2 where $P_{q_2} \sqsubseteq H_{q_1}$

SPARQL Query Containment & Equivalence I

Related Work

Abbas, A., Genevs, P., Roisin, C., and Layada, N. (2017).
SPARQL Query Containment with ShEx Constraints.
In *European Conference on Advances in Databases and Information Systems*, pages 343–356. Springer.

Chekol, M. W. (2016).
On the containment of SPARQL queries under entailment regimes.
In *Thirtieth AAAI Conference on Artificial Intelligence*.

Chekol, M. W., Euzenat, J., Genevs, P., and Layada, N. (2011).
PSPARQL query containment.
PhD Thesis, INRIA.

Chekol, M. W., Euzenat, J., Genevs, P., and Layada, N. (2012a).
A benchmark for semantic web query containment, equivalence and satisfiability.
PhD Thesis, INRIA.

Chekol, M. W., Euzenat, J., Genevs, P., and Layada, N. (2012b).
SPARQL query containment under RDFS entailment regime.
In *International Joint Conference on Automated Reasoning*, pages 134–148. Springer.

Chekol, M. W., Euzenat, J., Genevs, P., and Layada, N. (2012c).
SPARQL query containment under SHI axioms.
In *Twenty-Sixth AAAI Conference on Artificial Intelligence*.

Chekol, M. W., Euzenat, J., Genevs, P., and Layada, N. (2013).
Evaluating and benchmarking SPARQL query containment solvers.
In *International Semantic Web Conference*, pages 408–423. Springer.

SPARQL Query Containment & Equivalence II

Related Work

Chekol, M. W., Euzenat, J., Genevs, P., and Layada, N. (2018).
SPARQL query containment under schema.
Journal on Data Semantics, 7(3):133–154.

Kaminski, M. and Kostylev, E. V. (2019).
Subsumption of Weakly Well-Designed SPARQL Patterns is Undecidable.
arXiv preprint arXiv:1901.09353.

Pichler, R. and Skritek, S. (2014).
Containment and equivalence of well-designed SPARQL.
In *Proceedings of the 33rd ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 39–50. ACM.

ROUSSET, M. M.-C., Calvanese, M. D., Polleres, M. A., Mugnier, M. M.-L., Euzenat, M. J., and Layada, M. N. (2012).
Static Analysis of Semantic Web Queries.
PhD Thesis, University of Montpellier.

Salas, J. and Hogan, A. (2018a).
Canonicalisation of monotone SPARQL queries.
In *International Semantic Web Conference*, pages 600–616. Springer.

Salas, J. and Hogan, A. (2018b).
QCan: Normalising Congruent SPARQL Queries.
In *Proc. 17th International Semantic Web Conference (ISWC), pp. article*, volume 54.

Salas, J. and Hogan, A. (2019).
Canonicalisation of Monotone SPARQL Queries (Extended Version).

SPARQL Query Containment & Equivalence III

Related Work

Saleem, M., Stadler, C., Mehmood, Q., Lehmann, J., and Ngomo, A.-C. N. (2017).
SQCFramework: SPARQL query containment benchmark generation framework.
In *Proceedings of the Knowledge Capture Conference*, page 28. ACM.

Serfiotis, G., Koffina, I., Christophides, V., and Tannen, V. (2005).
Containment and minimization of RDF/S query patterns.
In *International Semantic Web Conference*, pages 607–623. Springer.

Stadler, C., Saleem, M., Ngomo, A.-C. N., and Lehmann, J. (2018).
Efficiently Pinpointing SPARQL Query Containments.
In Mikkonen, T., Klamma, R., and Hernndez, J., editors, *Web Engineering*, Lecture Notes in Computer Science, pages 210–224. Springer International Publishing.

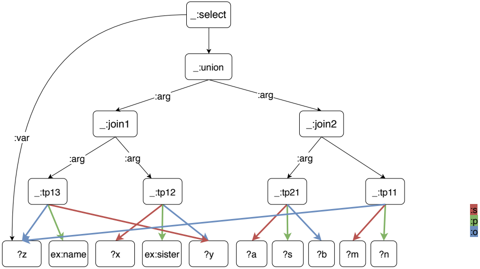
RDF Morphisms and SPARQL Query Containment

RDF Morphisms and SPARQL Query Containment

Representing SPARQL Queries as RDF Graphs

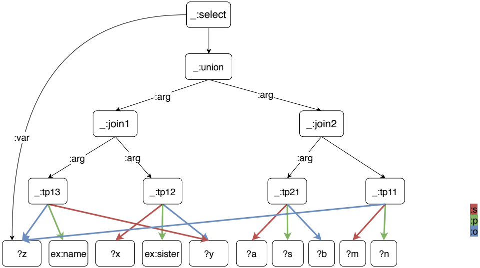
```
PREFIX : <http://dfki.de/voc#>

SELECT ?z
WHERE {
  { ?x :sister ?y .
    ?y :name ?z . }
  UNION
  { ?a ?s ?b .
    ?m ?n ?z . } }
```



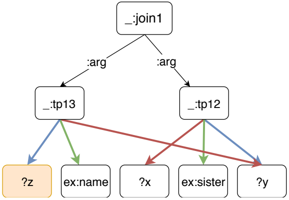
RDF Morphisms and SPARQL Query Containment

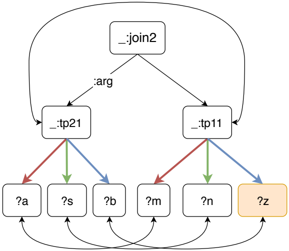
Algebraic Transformation into Union Normal Form



RDF Morphisms and SPARQL Query Containment

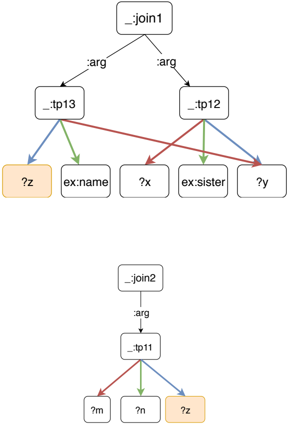
Computation of RDF cores for intra-dependencies removal in sub-CQs





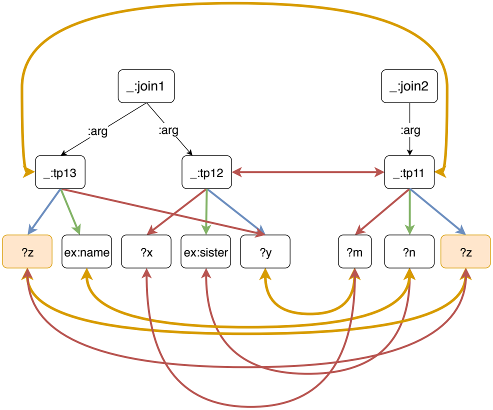
RDF Morphisms and SPARQL Query Containment

Computation of RDF cores for intra-dependencies removal in sub-CQs



RDF Morphisms and SPARQL Query Containment

Computation of subgraph isomorphisms for inter-dependencies removal in sub-CQs



RDF Morphisms and SPARQL Query Containment

Computation of subgraph isomorphisms for inter-dependencies removal in sub-CQs

```
PREFIX : <http://dfki.de/voc#>

SELECT ?z
WHERE {
  { ?x :sister ?y .
    ?y :name ?z . }
  UNION
  { ?a ?s ?b .
    ?m ?n ?z . } }
```