



TECHNISCHE
UNIVERSITÄT
DRESDEN

FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

SPARQL Entailment Regimes

Sebastian Rudolph

Dresden, June 18

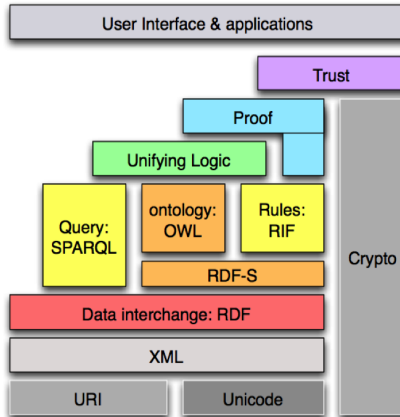


DRESDEN
SCHOOL OF
INFORMATICS
www.soi.tu-dresden.de

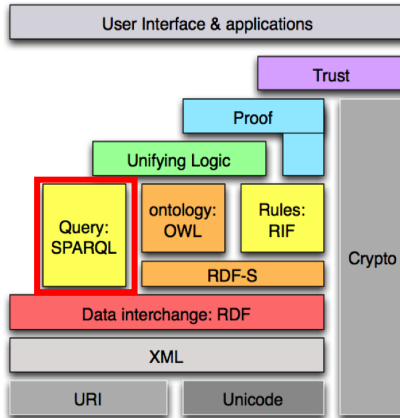
Content

Overview & XML	9 APR DS2	Hypertableau II	7 JUN DS5
Introduction into RDF	9 APR DS3	Tutorial 5	11 JUN DS2
RDFS – Syntax & Intuition	12 APR DS5	SPARQL Syntax & Intuition	11 JUN DS3
RDFS – Semantics	16 APR DS2	SPARQL – Semantics	14 JUN DS2
RDFS Rule-based Reasoning	16 APR DS3	SPARQL Algebra	14 JUN DS3
Tutorial 1	19 APR DS5	SPARQL 1.1	14 JUN DS5
OWL – Syntax & Intuition	26 APR DS5	Tutorial 6	18 JUN DS2
Tutorial 2	3 MAY DS5	SPARQL Entailment	18 JUN DS3
OWL – Syntax & Intuition ctd.	7 MAY DS2	Ontology Editing	2 JUL DS2
OWL & Description Logics	7 MAY DS3	Ontology Engineering	2 JUL DS3
OWL 2	10 MAY DS5	Tutorial 7	9 JUL DS2
Tutorial 3	14 MAY DS2	Linked Data	9 JUL DS3
Tableau I	14 MAY DS3	Applications	12 JUL DS5
Tableau II	17 MAY DS5	Test Exam	16 JUL DS2
Tutorial 4	7 JUN DS2	Test Exam Evaluation	16 JUL DS3
Hypertableau I	7 JUN DS3	Q&A Session	19 JUL DS5

The SPARQL Query Language



The SPARQL Query Language



Agenda

- 1 Introduction and Motivation
- 2 Conditions for Extending the Bgp Operator
- 3 BGP Evaluation with RDFS Entailment
- 4 Implementation Options
- 5 BGP Evaluation with OWL Semantics
- 6 Summary

Agenda

- 1 Introduction and Motivation
- 2 Conditions for Extending the Bgp Operator
- 3 BGP Evaluation with RDFS Entailment
- 4 Implementation Options
- 5 BGP Evaluation with OWL Semantics
- 6 Summary

Introduction and Motivation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- No answer using simple entailment/subgraph matching

SPARQL with Implicit Solutions

- So far: solutions through subgraph matching (simple entailment)
- Only the $Bgp(\cdot)$ algebra operator (exception: property paths) generates solutions
- SPARQL 1.0 specifies a BGP matching extension point to overwrite behaviour of $Bgp(\cdot)$

Idea: Instead of subgraph matching use entailment relations

Agenda

- 1 Introduction and Motivation
- 2 Conditions for Extending the Bgp Operator**
- 3 BGP Evaluation with RDFS Entailment
- 4 Implementation Options
- 5 BGP Evaluation with OWL Semantics
- 6 Summary

Previous BGP Evaluation

Definition (Solution)

Let P be a basic graph pattern. A partial function μ is a solution for $\text{Bgp}(P)$ over the queried (active) graph G if:

- 1 the domain of μ is exactly the set of variables in P ,
- 2 there exists an assignment σ from blank nodes in P to IRIs, blank nodes, or RDF literals in G such that:
- 3 the RDF graph $\mu(\sigma(P))$ is a subgraph of G .

The result $\llbracket \text{Bgp}(P) \rrbracket_G$ of the evaluation of $\text{Bgp}(P)$ over G is the multi set of solutions μ (multiplicity corresponds to the number of different assignments)

Naive Idea for BGP Evaluation using RDFS Entailment

Definition (Solution)

Let P be a basic graph pattern. A partial function μ is a solution for $\text{Bgp}(P)$ over the queried (active) graph G under RDFS entailment if:

- 1 the domain of μ is exactly the set of variables in P ,
- 2 there exists an assignment σ from blank nodes in P to IRIs, blank nodes, or RDF literals such that:
- 3 the RDF graph $\mu(\sigma(P))$ is RDFS-entailed by G .

The result $\llbracket \text{Bgp}(P) \rrbracket_G$ of the evaluation of $\text{Bgp}(P)$ over G under RDFS entailment is the multi set of such solutions

Conditions for Entailment Regimes (1)

- The naive idea produces not always intuitive results
- It is not that simple since such extensions have to satisfy several conditions

Conditions for Entailment Regimes (1)

- The naive idea produces not always intuitive results
- It is not that simple since such extensions have to satisfy several conditions

A so-called entailment regime E specifies

- 1 RDF Graphs that are well-formed for the regime
- 2 an entailment relation between well-formed graphs

Conditions for Entailment Regimes (1)

- The naive idea produces not always intuitive results
- It is not that simple since such extensions have to satisfy several conditions

A so-called entailment regime E specifies

- 1 RDF Graphs that are well-formed for the regime
- 2 an entailment relation between well-formed graphs

We can address this:

- 1 For RDF(S) all RDF graphs are ok, for OWL we will further define well-formed graphs
- 2 We can use already defined entailment relations

Conditions for Entailment Regimes (2)

An entailment regime E defines furthermore

3. The effect of a query over an inconsistent graph
4. Conditions to guarantee the uniqueness of the results modulo blank node labels

We can also address this:

3. Warning/error
4. Automatically satisfied for RDFS entailment

Conditions for Entailment Regimes (3)

An entailment regime E defines furthermore

5. Conditions such that for any basic graph pattern P and any graph G , if $\mu_1, \dots, \mu_n \in \llbracket P \rrbracket_G^E$ and P_1, \dots, P_n are copies of P not sharing any blank nodes with G or with each other:
$$G \models^E (G \cup \mu_1(P_1) \cup \dots \cup \mu_n(P_n))$$
6. Condition to prevent trivial infinite solutions

Condition 5 makes sure that blank nodes in solutions correspond to blank nodes in the graph (no unintended co-references are introduced)

Comment for Condition 5

Example

G : :a :b _ :c . G_1 : :a :b _ :b1 . G_2 : :a :b _ :b2 . G_3 : :a :b _ :b1 .
_ :d :e :f . _ :b2 :e :f . _ :b1 :e :f . _ :b1 :e :f .

- G has as simple consequences G_1 and G_2 , but not G_3 (blank nodes are merged)

Comment for Condition 5

Example

$G: :a :b _ :c . \quad G_1: :a :b _ :b1 . \quad G_2: :a :b _ :b2 . \quad G_3: :a :b _ :b1 .$
 $_ :d :e :f . \quad _ :b2 :e :f . \quad _ :b1 :e :f . \quad _ :b1 :e :f .$

- G has as simple consequences G_1 and G_2 , but not G_3 (blank nodes are merged)
- Let $P = \{ :a :b ?x . ?y :e :f \}$. We would have $\mu_1: ?x \mapsto _ :b1, ?y \mapsto _ :b2$ and $\mu_2: ?x \mapsto _ :b2, ?y \mapsto _ :b1$ as solutions for P over G since $\mu_1(P) = G_1, \mu_2(P) = G_2$

Comment for Condition 5

Example

$G: :a :b _ :c . \quad G_1: :a :b _ :b1 . \quad G_2: :a :b _ :b2 . \quad G_3: :a :b _ :b1 .$
 $_ :d :e :f . \quad _ :b2 :e :f . \quad _ :b1 :e :f . \quad _ :b1 :e :f .$

- G has as simple consequences G_1 and G_2 , but not G_3 (blank nodes are merged)
- Let $P = \{ :a :b ?x . ?y :e :f \}$. We would have $\mu_1: ?x \mapsto _ :b1, ?y \mapsto _ :b2$ and $\mu_2: ?x \mapsto _ :b2, ?y \mapsto _ :b1$ as solutions for P over G since $\mu_1(P) = G_1, \mu_2(P) = G_2$
- But $G \cup \mu_1(P) \cup \mu_2(P)$ is not a consequence (contains G_3)

Comment for Condition 5

Example

$G: :a :b _ :c . \quad G_1: :a :b _ :b1 . \quad G_2: :a :b _ :b2 . \quad G_3: :a :b _ :b1 .$
 $_ :d :e :f . \quad _ :b2 :e :f . \quad _ :b1 :e :f . \quad _ :b1 :e :f .$

- G has as simple consequences G_1 and G_2 , but not G_3 (blank nodes are merged)
- Let $P = \{ :a :b ?x . ?y :e :f \}$. We would have $\mu_1: ?x \mapsto _ :b1, ?y \mapsto _ :b2$ and $\mu_2: ?x \mapsto _ :b2, ?y \mapsto _ :b1$ as solutions for P over G since $\mu_1(P) = G_1, \mu_2(P) = G_2$
- But $G \cup \mu_1(P) \cup \mu_2(P)$ is not a consequence (contains G_3)
- Problem: we introduced unintended co-references

Agenda

- 1 Introduction and Motivation
- 2 Conditions for Extending the Bgp Operator
- 3 BGP Evaluation with RDFS Entailment**
- 4 Implementation Options
- 5 BGP Evaluation with OWL Semantics
- 6 Summary

Problems with the Naive Evaluation Idea (1)

Even an empty RDF Graph RDFS-entails infinitely many axiomatic triples:

- $\{\} \models_{\text{RDFS}} \text{rdf_}i \text{ rdf:type rdf:Property}$ for all $i \in \mathbf{N}$

Query

```
SELECT ?x WHERE { ?x rdf:type rdf:Property }
```

↪ Query has infinitely many solutions under RDFS entailment

Solution (1)

- Bindings are limited to a finite vocabulary

Definition (Solution)

Let P be a basic graph pattern. A partial function μ is a solution for $\text{Bgp}(P)$ over the queried (active) graph G under RDFS entailment if:

- 1 the domain of μ is exactly the set of variables in P ,
- 2 terms in the range of μ occur in G ,
- 3 there exists an assignment σ from blank nodes in P to IRIs, blank nodes, or RDF literals in G such that:
- 4 the RDF graph $\mu(\sigma(P))$ is RDFS-entailed by G .

Problem with the Naive Evaluation Idea (2)

Taking only the vocabulary of G is too strict:

- $\{ \text{ex:s ex:p ex:o . ex:p rdfs:domain ex:C } \models_{\text{RDFS}} \{ \text{ex:s rdf:type ex:C } \}$

Query

```
SELECT ?x WHERE { ex:s ?x ex:C }
```

Has no solutions ($\text{rdf:type} \notin \text{Voc}(G)$).

Solution (2)

- Let $\text{Voc}^-(\text{RDFS}) = \text{Voc}(\text{RDFS}) \setminus \{\text{rdf} : _i \mid i \in \mathbf{N}\}$

Definition (Solution)

Let P be a basic graph pattern. A partial function μ is a solution for $\text{Bgp}(P)$ over the queried (active) graph G under RDFS entailment if:

- 1 the domain of μ is exactly the set of variables in P ,
- 2 terms in the range of μ occur in G or $\text{Voc}^-(\text{RDFS})$,
- 3 there exists an assignment σ from blank nodes in P to IRIs, blank nodes, or RDF literals in G such that:
- 4 the RDF graph $\mu(\sigma(P))$ is RDFS-entailed by G .

Problems with the Naive Evaluation Idea (3)

Blank nodes have existential semantics

- $\{ \text{ex:s ex:p ex:o} \} \models_{\text{RDFS}} \{ \text{ex:s ex:p _ :id} \}$
for each id

Problems with the Naive Evaluation Idea (3)

Blank nodes have existential semantics

- $\{ \text{ex:s ex:p ex:o} \} \models_{\text{RDFS}} \{ \text{ex:s ex:p _ :id} \}$
for each id

We already guarantee finite results since the possible range of μ and σ is finite, but ...

Problems with the Naive Evaluation Idea (3)

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data

```
 $G_1 = \{ \text{ex:s1 ex:p1 \_ :a } \}$ 
```

```
 $G_2 = \{ \text{ex:s1 ex:p1 \_ :a . ex:s2 ex:p2 \_ :b } \}$ 
```

Problems with the Naive Evaluation Idea (3)

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data

```
 $G_1 = \{ \text{ex:s1 ex:p1 \_ :a } \}$ 
```

```
 $G_2 = \{ \text{ex:s1 ex:p1 \_ :a . ex:s2 ex:p2 \_ :b } \}$ 
```

- Has 1 solution for G_1 and 2 solutions for G_2

Problems with the Naive Evaluation Idea (3)

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data

```
 $G_1 = \{ \text{ex:s1 ex:p1 \_ :a } \}$ 
```

```
 $G_2 = \{ \text{ex:s1 ex:p1 \_ :a . ex:s2 ex:p2 \_ :b } \}$ 
```

- Has 1 solution for G_1 and 2 solutions for G_2
- Adding a triple that is unrelated to the first one causes new solutions

Problems with the Naive Evaluation Idea (3)

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data

$G_1 = \{ \text{ex:s1 ex:p1 _ :a } \}$

$G_2 = \{ \text{ex:s1 ex:p1 _ :a . ex:s2 ex:p2 _ :b } \}$

- Has 1 solution for G_1 and 2 solutions for G_2
- Adding a triple that is unrelated to the first one causes new solutions
- Solution: Skolemisation

Skolemisation

- Skolemisation: we consider the blank nodes as constants/normal IRIs

Definition (Skolemisation)

Let the prefix `skol` refer to a namespace IRI that does not occur as the prefix of any IRI in the queried (active) graph or query. The Skolemisation $sk(_:b)$ of a blank node `_:b` is defined as $sk(_:b) = skol:b$. We extend $sk(\cdot)$ to graphs in the natural way.

Example: Skolemisation

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data (Skolemised)

```
sk(G1) = { ex:s1 ex:p1 skol:a }
```

```
sk(G2) = { ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }
```

Example: Skolemisation

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data (Skolemised)

```
sk(G1) = { ex:s1 ex:p1 skol:a }
```

```
sk(G2) = { ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }
```

$sk(G_1) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$

$sk(G_1) \not\models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$

$sk(G_2) \models_{RDFS}^? \{ ex:s2 \ ex:p2 \ skol:b \}$

Example: Skolemisation

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data (Skolemised)

```
sk(G1) = { ex:s1 ex:p1 skol:a }
```

```
sk(G2) = { ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }
```

$sk(G_1) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$ ✓

$sk(G_1) \not\models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$

$sk(G_2) \models_{RDFS}^? \{ ex:s2 \ ex:p2 \ skol:b \}$

Example: Skolemisation

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data (Skolemised)

```
sk(G1) = { ex:s1 ex:p1 skol:a }
```

```
sk(G2) = { ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }
```

$sk(G_1) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$ ✓

$sk(G_1) \not\models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$ ✗

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$

$sk(G_2) \models_{RDFS}^? \{ ex:s2 \ ex:p2 \ skol:b \}$

Example: Skolemisation

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data (Skolemised)

```
sk(G1) = { ex:s1 ex:p1 skol:a }
```

```
sk(G2) = { ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }
```

$sk(G_1) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$ ✓

$sk(G_1) \not\models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$ ✗

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$ ✓

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$

$sk(G_2) \models_{RDFS}^? \{ ex:s2 \ ex:p2 \ skol:b \}$

Example: Skolemisation

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data (Skolemised)

```
sk(G1) = { ex:s1 ex:p1 skol:a }
```

```
sk(G2) = { ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }
```

$sk(G_1) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$ ✓

$sk(G_1) \not\models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$ ✗

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$ ✓

$sk(G_2) \not\models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$ ✗

$sk(G_2) \models_{RDFS}^? \{ ex:s2 \ ex:p2 \ skol:b \}$

Example: Skolemisation

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data (Skolemised)

```
sk(G1) = { ex:s1 ex:p1 skol:a }
```

```
sk(G2) = { ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }
```

$sk(G_1) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$ ✓

$sk(G_1) \not\models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$ ✗

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$ ✓

$sk(G_2) \not\models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$ ✗

$sk(G_2) \models_{RDFS}^? \{ ex:s2 \ ex:p2 \ skol:b \}$ ✓

Example: Skolemisation

Query

```
SELECT ?x WHERE { ex:s1 ex:p1 ?x }
```

Data (Skolemised)

```
sk(G1) = { ex:s1 ex:p1 skol:a }
```

```
sk(G2) = { ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }
```

$sk(G_1) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$ ✓

$sk(G_1) \not\models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$ ✗

$sk(G_2) \models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:a \}$ ✓

$sk(G_2) \not\models_{RDFS}^? \{ ex:s1 \ ex:p1 \ skol:b \}$ ✗

$sk(G_2) \models_{RDFS}^? \{ ex:s2 \ ex:p2 \ skol:b \}$ ✓

Only 1 Solution $\mu: ?x \mapsto skol:a$ for $sk(G_1)$ and $sk(G_2)$

Problems with Skolemisation

- Of course we do not want to see Skolem constants in solutions
- ↪ Use Skolemisation only as a condition, applied to the graph and query

Solutions in the RDFS Entailment Regime

Definition (Solutions under RDFS entailment)

Let P be a basic graph pattern. A partial function μ is a solution for $\text{Bgp}(P)$ over the queried (active) graph G under RDFS entailment if:

- 1 the domain of μ is exactly the set of variables in P ,
- 2 terms in the range of μ occur in G or Voc^- (RDFS),
- 3 there exists an assignment σ from blank nodes in P to IRIs, blank nodes, or RDF literals in G such that:
- 4 the RDF graph $\text{sk}(\mu(\sigma(P)))$ is well-formed and RDFS-entailed by G .

The well-formed criterion prevents literals in subject position

SPARQL Entailment Regime

SPARQL entailment regimes define

- A name for the regime
- What entailment relation is used, e.g., RDFS-entailment
- Above described restrictions to address extension point conditions
- Legal graphs and queries (for RDFS all RDF graphs and SPARQL queries are legal)
- Handling of inconsistencies
- Errors handling
- How a regime can be described in SPARQL service descriptions

Standard SPARQL Semantics as Entailment Regime

Definition (Solutions under **simple** entailment)

Let P be a basic graph pattern. A partial function μ is a solution for $\text{Bgp}(P)$ over the queried (active) graph G under ~~RDFS~~ **simple entailment** if:

- 1 the domain of μ is exactly the set of variables in P ,
- 2 terms in the range of μ occur in G or ~~$\text{Voc}^-(\text{RDFS})$~~ ,
- 3 there exists an assignment σ from blank nodes in P to IRIs, blank nodes, or RDF literals in G such that:
- 4 the RDF graph $\text{sk}(\mu(\sigma(P)))$ is well-formed and ~~RDFS~~ **simple** entailed by G .

Standard SPARQL Semantics as Entailment Regime

Definition (Solutions under **simple** entailment)

Let P be a basic graph pattern. A partial function μ is a solution for $\text{Bgp}(P)$ over the queried (active) graph G under ~~RDFS~~ **simple entailment** if:

- 1 the domain of μ is exactly the set of variables in P ,
- 2 terms in the range of μ occur in G or ~~$\text{Voc}^-(\text{RDFS})$~~ ,
- 3 there exists an assignment σ from blank nodes in P to IRIs, blank nodes, or RDF literals in G such that:
- 4 the RDF graph $\text{sk}(\mu(\sigma(P)))$ is well-formed and ~~RDFS~~ **simple** entailed by G .

↪ Same definition can be used with simple entailment to obtain subgraph matching semantics

Agenda

- 1 Introduction and Motivation
- 2 Conditions for Extending the Bgp Operator
- 3 BGP Evaluation with RDFS Entailment
- 4 Implementation Options**
- 5 BGP Evaluation with OWL Semantics
- 6 Summary

Implementation of the RDFS Entailment Regime

The definition based on entailment relations allows for different implementation techniques

- Materialisation / forwards-chaining
- Query rewriting / backwards-chaining
- Hybrid approaches

RDFS Entailment Regime via Materialisation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- No answer under simple entailment/subgraph matching

RDFS Entailment Regime via Materialisation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- No answer under simple entailment/subgraph matching
- Idea: we extend the queried graph with relevant inferred triples

RDFS Entailment Regime via Materialisation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

RDFS Entailment Regime via Materialisation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

RDFS Entailment Regime via Materialisation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .  
ex:Birte rdf:type ex:Lecturer .
```

RDFS Entailment Regime via Materialisation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .  
ex:Birte rdf:type ex:Lecturer .
```

RDFS Entailment Regime via Materialisation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .  
ex:Birte rdf:type ex:Lecturer .  
ex:Birte rdf:type ex:Person .
```

RDFS Entailment Regime via Materialisation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .  
ex:Birte rdf:type ex:Lecturer .  
ex:Birte rdf:type ex:Person .
```

- Query over the extended graph: $\mu: ?x \mapsto \text{ex:Birte}$

RDFS Entailment Regime via Materialisation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .  
ex:Birte rdf:type ex:Lecturer .  
ex:Birte rdf:type ex:Person .
```

- Query over the extended graph: $\mu: ?x \mapsto \text{ex:Birte}$
- Disadvantages:

RDFS Entailment Regime via Materialisation

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .  
ex:Birte rdf:type ex:Lecturer .  
ex:Birte rdf:type ex:Person .
```

- Query over the extended graph: $\mu: ?x \mapsto \text{ex:Birte}$
- Disadvantages:
 - Size of the queried graph grows
 - Each update requires recomputation of the closure (extension)

RDFS Ent. Regime via Query Rewriting

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- Idea: extend the query rather than the queried graph

RDFS Ent. Regime via Query Rewriting

Query

```
SELECT ?x WHERE { ?x a ex:Person }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- Idea: extend the query rather than the queried graph
- Rule rdfs9 produces a relevant consequence

$$\frac{u \text{ rdfs:subClassOf } x . \quad v \text{ rdf:type } u .}{v \text{ rdf:type } x .} \text{ rdfs9}$$

RDFS Ent. Regime via Query Rewriting

Query

```
SELECT ?x WHERE { ?x a ex:Person } UNION  
                 { ?x a ex:Lecturer }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- Idea: extend the query rather than the queried graph
- Rule rdfs9 produces a relevant consequence

$$\frac{u \text{ rdfs:subClassOf } x . \quad v \text{ rdf:type } u .}{v \text{ rdf:type } x .} \text{ rdfs9}$$

RDFS Ent. Regime via Query Rewriting

Query

```
SELECT ?x WHERE { ?x a ex:Person } UNION  
                 { ?x a ex:Lecturer }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- Rule rdfs2 produces now also a relevant consequence

$$\frac{a \text{ rdfs:domain } x . \quad u \text{ a } y .}{u \text{ rdf:type } x .} \text{ rdfs2}$$

RDFS Ent. Regime via Query Rewriting

Query

```
SELECT ?x WHERE { ?x a ex:Person } UNION  
                 { ?x a ex:Lecturer } UNION  
                 { ?x ex:presentsLecture _:y }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- Rule rdfs2 produces now also a relevant consequence

$$\frac{a \text{ rdfs:domain } x . \quad u \text{ a } y .}{u \text{ rdf:type } x .} \text{ rdfs2}$$

RDFS Ent. Regime via Query Rewriting

Query

```
SELECT ?x WHERE { ?x a ex:Person } UNION  
                 { ?x a ex:Lecturer } UNION  
                 { ?x ex:presentsLecture _:y }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- Solution $\mu: ?x \mapsto ex: Birte$ (from 3. disjunct)

RDFS Ent. Regime via Query Rewriting

Query

```
SELECT ?x WHERE { ?x a ex:Person } UNION  
                 { ?x a ex:Lecturer } UNION  
                 { ?x ex:presentsLecture _:y }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- Solution $\mu: ?x \mapsto ex: Birte$ (from 3. disjunct)
- Disadvantages:

RDFS Ent. Regime via Query Rewriting

Query

```
SELECT ?x WHERE { ?x a ex:Person } UNION  
                 { ?x a ex:Lecturer } UNION  
                 { ?x ex:presentsLecture _:y }
```

Data

```
ex:Birte ex:presentsLecture "SPARQL" .  
ex:presentsLecture rdfs:domain ex:Lecturer .  
ex:Lecturer rdfs:subClassOf ex:Person .
```

- Solution $\mu: ?x \mapsto ex: Birte$ (from 3. disjunct)
- Disadvantages:
 - Hard/impossible to find all solutions (RDFS vocabulary used in unusual ways, queries not just for instances or subclasses)
 - Query Rewriting is done at run-time \rightsquigarrow every query is evaluated a bit slower

Hybrid Approaches

- Combine materialisation and query rewriting
- Common (beyond RDFS): do not materialise `owl:sameAs`
- Extract schema part and use that for rewriting

Agenda

- 1 Introduction and Motivation
- 2 Conditions for Extending the Bgp Operator
- 3 BGP Evaluation with RDFS Entailment
- 4 Implementation Options
- 5 BGP Evaluation with OWL Semantics**
- 6 Summary

SPARQL with OWL Direct Semantics

How can we use OWL's Direct Semantics with SPARQL?

- 1 Based on Description Logics
- 2 Semantics defined in terms of OWL structural objects
 - `owl:intersectionOf`, `ObjectIntersectionOf`, \sqcap
- 3 OWL DL ontologies can be mapped into RDF graphs
- 4 Not every RDF graph can be mapped into an OWL DL ontology

SPARQL with OWL Direct Semantics

- 1 OWL Direct Semantics Entailment Regime only works on **well-formed** RDF graphs, which can be mapped into OWL DL ontologies

SPARQL with OWL Direct Semantics

- 1 OWL Direct Semantics Entailment Regime only works on **well-formed** RDF graphs, which can be mapped into OWL DL ontologies
- 2 Basic graph patterns are mapped into **extended** OWL structural objects with variables

SPARQL with OWL Direct Semantics

- 1 OWL Direct Semantics Entailment Regime only works on **well-formed** RDF graphs, which can be mapped into OWL DL ontologies
- 2 Basic graph patterns are mapped into **extended** OWL structural objects with variables
- 3 Type declarations required to disambiguate the parsing process
 - `?x rdfs:subPropertyOf ?y .`

SPARQL with OWL Direct Semantics

- 1 OWL Direct Semantics Entailment Regime only works on **well-formed** RDF graphs, which can be mapped into OWL DL ontologies
- 2 Basic graph patterns are mapped into **extended** OWL structural objects with variables
- 3 Type declarations required to disambiguate the parsing process
 - `?x rdfs:subPropertyOf ?y .`
 - `?x a owl:ObjectProperty .`
 - `?y a owl:ObjectProperty .`

SPARQL with OWL Direct Semantics

- 1 OWL Direct Semantics Entailment Regime only works on **well-formed** RDF graphs, which can be mapped into OWL DL ontologies
- 2 Basic graph patterns are mapped into **extended** OWL structural objects with variables
- 3 Type declarations required to disambiguate the parsing process
 - `?x rdfs:subPropertyOf ?y .`
 - `?x a owl:ObjectProperty .`
 - `?y a owl:ObjectProperty .`
- 4 Variables can occur in class, property, individual, or literal positions

SPARQL with OWL Direct Semantics

- 1 OWL Direct Semantics Entailment Regime only works on **well-formed** RDF graphs, which can be mapped into OWL DL ontologies
- 2 Basic graph patterns are mapped into **extended** OWL structural objects with variables
- 3 Type declarations required to disambiguate the parsing process
 - `?x rdfs:subPropertyOf ?y .`
 - `?x a owl:ObjectProperty .`
 - `?y a owl:ObjectProperty .`
- 4 Variables can occur in class, property, individual, or literal positions
- 5 Definition of solutions analogously to the one for RDFS plus specification of well-formed BGP's and graphs

Implementation of the OWL DS Regime

- Materialisation impossible
- For example, we could have arbitrary disjunctions in the query (e.g., matching students that are not profs):

```
SELECT ?x WHERE { ?x a [ a owl:Class ;  
owl:ObjectUnionOf ( ex:Student ex:Prof ) ] }
```

Implementation of the OWL DS Regime

- Materialisation impossible
 - For example, we could have arbitrary disjunctions in the query (e.g., matching students that are not profs):

```
SELECT ?x WHERE { ?x a [ a owl:Class ;  
  owl:ObjectUnionOf ( ex:Student ex:Prof ) ] }
```
 - Turtle is not an easy syntax for complex OWL expressions
 ↪ Usability problems
 - Queries go beyond simple instance queries
 - Optimisation is difficult for such complex queries
- ↪ Often we have to test all possible bindings

SPARQL with OWL Profiles

OWL Profiles better suited for web applications

- OWL RL profile can be implemented via materialisation
- Polynomial complexity
- Extends RDFS semantics (i.e., can be used with OWL's RDF-Based Semantics)
- Works on arbitrary RDF graphs

Further Entailment Regimes

- RDF Entailment Regime (just simpler than RDFS)
- D-Entailment Regime (adds datatype reasoning to RDFS)
- RIF Core Entailment Regime
 - Specify rules and query an RDF graph plus the rules

Agenda

- 1 Introduction and Motivation
- 2 Conditions for Extending the Bgp Operator
- 3 BGP Evaluation with RDFS Entailment
- 4 Implementation Options
- 5 BGP Evaluation with OWL Semantics
- 6 Summary**

Summary

- SPARQL can now be used with RDF(S), OWL, and RIF semantics
- Entailment Regimes overwrite evaluation of basic graph patterns
- Property Paths from SPARQL Query 1.1 problematic
- Definition of solutions (relatively) general
 - Works also for subgraph matching/simple entailment
 - OWL's Direct Semantics needs extra conditions/definitions
- Implementation and efficiency for OWL problematic
 - ↔ OWL 2 Profiles