TECHNISCHE UNIVERSITÄT DRESDEN

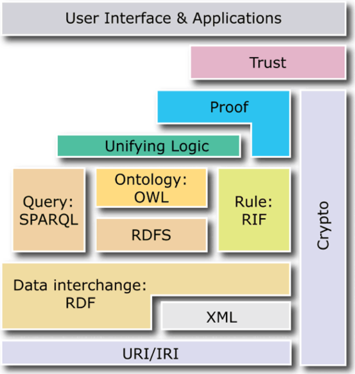
FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

SPARQL Entailment Regimes

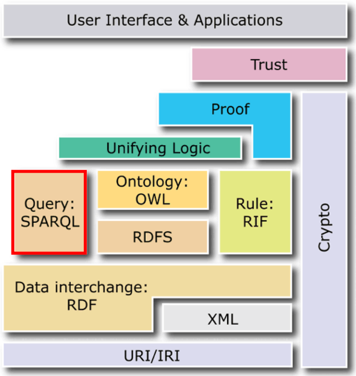
Sebastian Rudolph

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

The SPARQL Query Language



The SPARQL Query Language



Agenda

Agenda

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(·) algebra operator (exception: property paths) generates solutions
- SPARQL 1.0 specifies a BGP matching extension point to overwrite behaviour of Bgp(·)

Idea: Instead of subgraph matching use entailment relations

Agenda

TECHNISCHE UNIVERSITÄT DRESDEN

Previous BGP Evaluation

Definition (Solution)

Let P be a basic graph pattern. A partial function μ is a solution for 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 $[\![\text{Bgp}(P)]\!]_G$ of the evaluation of Bgp(P) over G is the multi set of solutions μ (multiplicity corresponds to the number of different assignments)

TECHNISCHE UNIVERSITÄT DRESDEN

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 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 $[\![\text{Bgp}(P)]\!]_G$ of the evaluation of 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 $\mu_1, \ldots, \mu_n \in [\![P]\!]_{\mathsf{G}}^{\mathsf{E}}$ and $P_1, \ldots, P_n$ are copies of P not sharing any blank nodes with G or with each other:
   $G \models^E (G \cup \mu_1(P_1) \cup \ldots \cup \mu_n(P_n))$
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 .`
`_:d :e :f .`

G_1: `:a :b _:b1 .`
`_:b2 :e :f .`

G_2: `:a :b _:b2 .`
`_:b1 :e :f .`

G_3: `:a :b _:b1 .`
`_: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 .` `_:d :e :f .` G_1: `:a :b _:b1 .` `_:b2 :e :f .` G_2: `:a :b _:b2 .` `_:b1 :e :f .` G_3: `:a :b _:b1 .` `_: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 μ_1: `?x` $\mapsto$ `_:b1`, `?y` $\mapsto$ `_:b2` and μ_2: `?x` $\mapsto$ `_:b2`, `?y` $\mapsto$ `_:b1` as solutions for `P` over G since $\mu_1(\texttt{P}) = G_1$, $\mu_2(\texttt{P}) = G_2$

Comment for Condition 5

Example

G: `:a :b _:c .`
`_:d :e :f .`

G_1: `:a :b _:b1 .`
`_:b2 :e :f .`

G_2: `:a :b _:b2 .`
`_:b1 :e :f .`

G_3: `:a :b _:b1 .`
`_: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 μ_1: `?x` $\mapsto$ `_:b1`, `?y` $\mapsto$ `_:b2` and μ_2: `?x` $\mapsto$ `_:b2`, `?y` $\mapsto$ `_:b1` as solutions for `P` over G since $\mu_1(\texttt{P}) = G_1$, $\mu_2(\texttt{P}) = G_2$
- But $G \cup \mu_1(\texttt{P}) \cup \mu_2(\texttt{P})$ is not a consequence (contains G_3)

Comment for Condition 5

Example

G: `:a :b _:c .` `_:d :e :f .`

G_1: `:a :b _:b1 .` `_:b2 :e :f .`

G_2: `:a :b _:b2 .` `_:b1 :e :f .`

G_3: `:a :b _:b1 .` `_: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 μ_1: `?x` $\mapsto$ `_:b1`, `?y` $\mapsto$ `_:b2` and μ_2: `?x` $\mapsto$ `_:b2`, `?y` $\mapsto$ `_:b1` as solutions for `P` over G since $\mu_1(\mathtt{P}) = G_1$, $\mu_2(\mathtt{P}) = G_2$
- But $G \cup \mu_1(\mathtt{P}) \cup \mu_2(\mathtt{P})$ is not a consequence (contains G_3)
- Problem: we introduced unintended co-references

Agenda

Problems with the Naive Evaluation Idea (1)

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

- $\{\} \models_{\text{RDFS}}$ `rdf:_i rdf:type rdf:Property` for all $i \in \mathbb{N}$

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 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:

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

Query

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

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

Solution (2)

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

Definition (Solution)

Let P be a basic graph pattern. A partial function μ is a solution for 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

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

Problems with the Naive Evaluation Idea (3)

Blank nodes have existential semantics

- `{ ex:s ex:p ex:o }` $\models_{\text{RDFS}}$ `{ 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 = { ex:s1 ex:p1 _:a }
G_2 = { 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 = `{ ex:s1 ex:p1 _:a }`
G_2 = `{ 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 = `{ ex:s1 ex:p1 _:a }`
G_2 = `{ 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 = { ex:s1 ex:p1 _:a }
G_2 = { 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(·) to graphs in the natural way.

Example: Skolemisation

Query

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

Data (Skolemised)

$sk(G_1)$ = `{ ex:s1 ex:p1 skol:a }`
$sk(G_2)$ = `{ 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)

$sk(G_1)$ = `{ ex:s1 ex:p1 skol:a }`
$sk(G_2)$ = `{ ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }`

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

$sk(G_1) \models^{?}_{RDFS}$ `{ 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)

$sk(G_1)$ = `{ ex:s1 ex:p1 skol:a }`
$sk(G_2)$ = `{ ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }`

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

$sk(G_1) \models^?_{RDFS}$ `{ 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)

$sk(G_1)$ = { `ex:s1 ex:p1 skol:a` }
$sk(G_2)$ = { `ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b` }

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

$sk(G_1) \models^?_{RDFS}$ { `ex:s1 ex:p1 skol:b` } $\lightning$

$sk(G_2) \models^?_{RDFS}$ { `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)

$sk(G_1)$ = `{ ex:s1 ex:p1 skol:a }`
$sk(G_2)$ = `{ ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }`

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

$sk(G_1) \models^?_{RDFS}$ `{ 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)

$sk(G_1)$ = `{ ex:s1 ex:p1 skol:a }`
$sk(G_2)$ = `{ ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }`

$sk(G_1) \models^?_{RDFS}$ `{ ex:s1 ex:p1 skol:a }` ✓
$sk(G_1) \models^?_{RDFS}$ `{ ex:s1 ex:p1 skol:b }` $\lightning$
$sk(G_2) \models^?_{RDFS}$ `{ ex:s1 ex:p1 skol:a }` ✓
$sk(G_2) \models^?_{RDFS}$ `{ ex:s1 ex:p1 skol:b }` $\lightning$
$sk(G_2) \models^?_{RDFS}$ `{ ex:s2 ex:p2 skol:b }`

Example: Skolemisation

Query

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

Data (Skolemised)

$sk(G_1)$ = `{ ex:s1 ex:p1 skol:a }`
$sk(G_2)$ = `{ ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }`

$sk(G_1) \models^?_{RDFS}$ `{ ex:s1 ex:p1 skol:a }` ✓
$sk(G_1) \models^?_{RDFS}$ `{ ex:s1 ex:p1 skol:b }` $\lightning$
$sk(G_2) \models^?_{RDFS}$ `{ ex:s1 ex:p1 skol:a }` ✓
$sk(G_2) \models^?_{RDFS}$ `{ ex:s1 ex:p1 skol:b }` $\lightning$
$sk(G_2) \models^?_{RDFS}$ `{ ex:s2 ex:p2 skol:b }` ✓

Example: Skolemisation

Query

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

Data (Skolemised)

$sk(G_1)$ = `{ ex:s1 ex:p1 skol:a }`
$sk(G_2)$ = `{ ex:s1 ex:p1 skol:a . ex:s2 ex:p2 skol:b }`

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

$sk(G_1) \models^?_{RDFS}$ `{ ex:s1 ex:p1 skol:b }` $\lightning$

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

$sk(G_2) \models^?_{RDFS}$ `{ ex:s1 ex:p1 skol:b }` $\lightning$

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

Only 1 Solution μ: `?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

$\rightsquigarrow$ 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 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 sk($\mu(\sigma(\mathtt{P}))$) 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 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 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 sk($\mu(\sigma(\mathtt{P}))$) is well-formed and ~~RDFS~~ simply 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 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 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 sk$(\mu(\sigma(\mathtt{P})))$ is well-formed and ~~RDFS~~ simply entailed by G.

$\leadsto$ Same definition can be used with simple entailment to obtain subgraph matching semantics

Agenda

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: μ: `?x` $\mapsto$ `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: μ: `?x` $\mapsto$ `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: μ: `?x` $\mapsto$ `ex:Birte`
- Disadvantages:
  - Size of the queried graph grows
  - Each update requires recomputation of the closure (extension)

TECHNISCHE UNIVERSITÄT DRESDEN

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

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

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

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

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

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

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

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

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 μ: `?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 μ: `?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 μ: `?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

TECHNISCHE UNIVERSITÄT DRESDEN

Agenda

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 BGPs and graphs

TECHNISCHE UNIVERSITÄT DRESDEN

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
  $\leadsto$ Usability problems
- Queries go beyond simple instance queries
- Optimisation is difficult for such complex queries

$\leadsto$ 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

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
  $\rightsquigarrow$ OWL 2 Profiles