ulm university universität ulm

Semantic Web Grundlagen

SPARQL Entailment Regimes

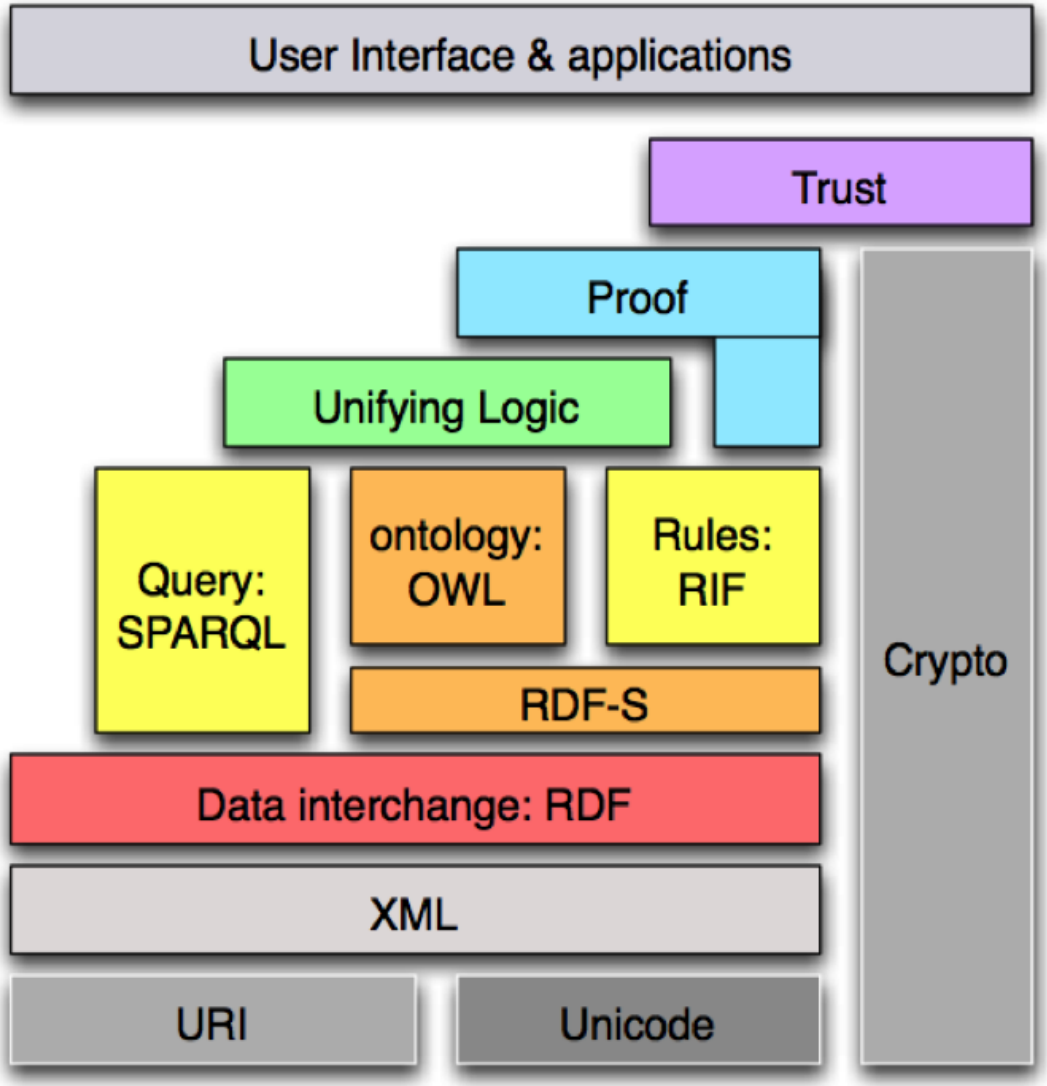
Birte Glimm
Institut für Künstliche Intelligenz | 16. Jan 2012

Organisatorisches: Inhalt

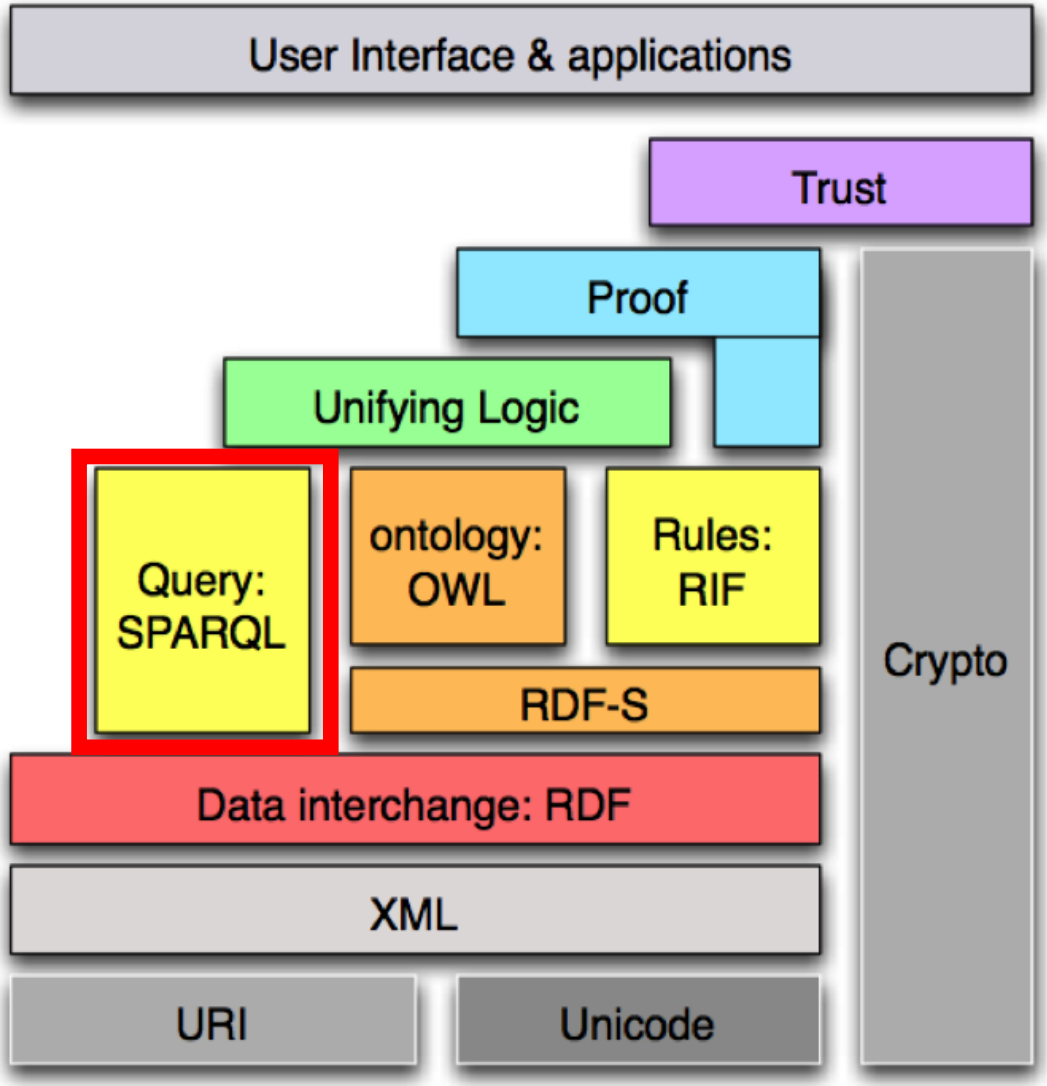
Einleitung und XML	17. Okt	Hypertableau II	12. Dez
Einführung in RDF	20. Okt	Übung 4	15. Dez
RDF Schema	24. Okt	SPARQL Syntax & Intuition	19. Dez
fällt aus	27. Okt	SPARQL Semantik	22. Dez
Logik – Grundlagen	31. Okt	SPARQL 1.1	9. Jan
Übung 1	3. Nov	Übung 5	12. Jan
Semantik von RDF(S)	7. Nov	SPARQL Entailment	16. Jan
RDF(S) & Datalog Regeln	10. Nov	SPARQL Implemetierung	19. Jan
OWL Syntax & Intuition	14. Nov	Ontology Editing	23. Jan
Übung 2	17. Nov	Übung 6	26. Jan
OWL & BLs	21. Nov	Ontology Engineering	30. Jan
OWL 2	24. Nov	Linked Data	2. Feb
Tableau	28. Nov	SemWeb Anwendungen	6. Feb
Übung 3	1. Dez	Übung 7	9. Feb
Blocking & Unravelling	5. Dez	Wiederholung	13. Feb
Hypertableau	8. Dez	Übung 8	16. Feb

Abfragen und RIF wurde gestrichen

Die Abfragesprache SPARQL



Die Abfragesprache SPARQL



Agenda

- Einleitung und Motivation
- Bedingungen für die Erweiterungen der Musterauswertung
- Musterauswertung mit RDFS Schlussfolgern
- Implementierungsoptionen
- Musterauswertung mit Unterstützung für Datentypen
- Musterauswertung mit OWL Schlussfolgern

Agenda

- Einleitung und Motivation
- Bedingungen für die Erweiterungen der Musterauswertung
- Musterauswertung mit RDFS Schlussfolgern
- Implementierungsoptionen
- Musterauswertung mit Unterstützung für Datentypen
- Musterauswertung mit OWL Schlussfolgern

Einführung und Motivation

Abfrage

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

Daten

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

- Keine Antwort unter einfachem Entailment

SPARQL mit Impliziten Lösungen

- Bisher: Lösungen durch Identifizierung von Teilgraphen
- Nur der Bgp(·) Algebra Operator (Ausnahme: Property Path) generiert Lösungen
- SPARQL 1.0 spezifiziert bereits einen Erweiterungspunkt für die Auswertung des Bgp(·) Operators

Idee: Wir nutzen statt Identifizierung von Teilgraphen Entailment Relationen zur Bestimmung der Lösungen

Agenda

- Einleitung und Motivation
- Bedingungen für die Erweiterungen der Musterauswertung
- Musterauswertung mit RDFS Schlussfolgern
- Implementierungsoptionen
- Musterauswertung mit Unterstützung für Datentypen
- Musterauswertung mit OWL Schlussfolgern

Bisherige Berechnung einfacher Graphmuster

Eine partielle Funktion μ ist eine Lösung des Ausdrucks *Bgp(P)* (P ein Muster) und des angefragten Graphen G, wenn:

1. Die Domäne von μ genau die Menge der Variablen in P ist
2. Es ein RDF Instanz Mapping σ von leeren Knoten in P zu URIs, leeren Knoten oder RDF-Literalen gibt, so dass gilt:

   Der Graph $\mu(\sigma(P))$ ist ein Teilgraph von G

Ergebnis der Auswertung von *Bgp(P)* über G, geschrieben $[\![\mathit{Bgp}(P)]\!]_G$ ist die Multimenge solcher Lösungen μ

- wobei die Vielfachheit jedes μ der Anzahl der verschiedenen RDF Instanz Mappings σ entspricht

Naiver Idee: Berechnung einfacher Graphmuster unter RDFS

Eine partielle Funktion μ ist eine Lösung des Ausdrucks *Bgp(P)* und des angefragten Graphen G (P: BGP) unter RDFS Entailment, wenn:

1. Die Domäne von μ genau die Menge der Variablen in P ist
2. Es ein RDF Instanz Mapping σ von leeren Knoten in P zu URIs, leeren Knoten oder RDF-Literalen gibt, so dass gilt:

   Der Graph $\mu(\sigma(P))$ ist RDFS-entailed von G.

Ergebnis der Auswertung von *Bgp(P)* über G, geschrieben $[\![Bgp(P)]\!]_G^{\text{RDFS}}$ ist die Multimenge solcher Lösungen μ

- wobei die Vielfachheit jedes μ der Anzahl der verschiedenen RDF Instanz Mappings σ entspricht

Bedingungen für Entailment Regimes (1)

- Der naive Vorschlag ist noch nicht immer intuitiv
- Ganz so einfach geht es auch nicht, denn derartige Erweiterungen müssen bestimmte Bedingungen erfüllen

Bedingungen für Entailment Regimes (1)

- Der naive Vorschlag ist noch nicht immer intuitiv
- Ganz so einfach geht es auch nicht, denn derartige Erweiterungen müssen bestimmte Bedingungen erfüllen

An entailment regime E specifies

1. a subset of RDF graphs called well-formed for the regime
2. an entailment relation between subsets of well-formed graphs and well-formed graphs

Bedingungen für Entailment Regimes (1)

- Der naive Vorschlag ist noch nicht immer intuitiv
- Ganz so einfach geht es auch nicht, denn derartige Erweiterungen müssen bestimmte Bedingungen erfüllen

An entailment regime E specifies

1. a subset of RDF graphs called well-formed for the regime
2. an entailment relation between subsets of well-formed graphs and well-formed graphs

Das können wir adressieren:

1. Für RDF(S) sind alle RDF Graphen ok, für OWL werden wir Wohlgeformtheit definieren
2. Hier können wir auf die bereits bekannten Schlussfolgerungsrelationen zurückgreifen

Bedingungen für Entailment Regimes (2)

An entailment regime E further specifies

- The effect of a query on an inconsistent graph must be specified by the particular SPARQL extension.
- An entailment regime E must provide conditions on basic graph pattern evaluation such that for any basic graph pattern P, any RDF graph G, and any evaluation that satisfies the conditions, the resulting multiset of solutions is uniquely determined up to RDF graph equivalence.

Bedingungen für Entailment Regimes (3)

An entailment regime must further satisfy:

1. For any consistent active graph G_a, the entailment regime E uniquely specifies a scoping graph G_s that is E-equivalent to G_a.
2. A set of well-formed graphs for E is specified such that, for any basic graph pattern P, scoping graph G_s, and solution mapping μ in $[\![P]\!]^{\mathsf{E}}_{\mathsf{G_s}}$, the graph $\mu(P)$ is well-formed for E.
3. For any basic graph pattern P and scoping graph G_s, if $\mu_1, \ldots, \mu_n$ in $[\![P]\!]^{\mathsf{E}}_{\mathsf{G_s}}$ and $P_1, \ldots, P_n$ are basic graph patterns all equivalent to P but not sharing any blank nodes with each other or with G_s, then $G_s \models^E (G_s \cup \mu_1(P_1) \cup \ldots \cup \mu_n(P_n))$
4. Entailment regimes should provide conditions to prevent trivial infinite solution multisets as appropriate to the regime.

Erläuterung zu Bedingung 3

Leere Knoten in Lösungen sollen leeren Knoten im Graphen entsprechen (keine unbeabsichtigten Koreferenzen):

Beispiel

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 hat als einfache Konsequenz G_1 und G_2, aber nicht G_3 (leere Knoten sind verschmolzen)

Erläuterung zu Bedingung 3

Leere Knoten in Lösungen sollen leeren Knoten im Graphen entsprechen (keine unbeabsichtigten Koreferenzen):

Beispiel

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 hat als einfache Konsequenz G_1 und G_2, aber nicht G_3 (leere Knoten sind verschmolzen)
- Sei `P = { :a :b ?x . ?y :e :f }`. Entsprechend wären μ_1: `?x` $\mapsto$ `_:b1`, `?y` $\mapsto$ `_:b2` und μ_2: `?x` $\mapsto$ `_:b2`, `?y` $\mapsto$ `_:b1` Lösungen für `P` über G, da $\mu_1(\texttt{P}) = G_1$, $\mu_2(\texttt{P}) = G_2$

Erläuterung zu Bedingung 3

Leere Knoten in Lösungen sollen leeren Knoten im Graphen entsprechen (keine unbeabsichtigten Koreferenzen):

Beispiel

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 hat als einfache Konsequenz G_1 und G_2, aber nicht G_3 (leere Knoten sind verschmolzen)
- Sei `P = { :a :b ?x . ?y :e :f }`. Entsprechend wären μ_1: `?x` $\mapsto$ `_:b1`, `?y` $\mapsto$ `_:b2` und μ_2: `?x` $\mapsto$ `_:b2`, `?y` $\mapsto$ `_:b1` Lösungen für `P` über G, da $\mu_1(\texttt{P}) = G_1$, $\mu_2(\texttt{P}) = G_2$
- Aber $G \cup \mu_1(\texttt{P}) \cup \mu_2(\texttt{P})$ ist keine Konsequenz (enthält G_3)

Erläuterung zu Bedingung 3

Leere Knoten in Lösungen sollen leeren Knoten im Graphen entsprechen (keine unbeabsichtigten Koreferenzen):

Beispiel

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 hat als einfache Konsequenz G_1 und G_2, aber nicht G_3 (leere Knoten sind verschmolzen)
- Sei `P = { :a :b ?x . ?y :e :f }`. Entsprechend wären μ_1: `?x` $\mapsto$ `_:b1`, `?y` $\mapsto$ `_:b2` und μ_2: `?x` $\mapsto$ `_:b2`, `?y` $\mapsto$ `_:b1` Lösungen für `P` über G, da $\mu_1(\texttt{P}) = G_1$, $\mu_2(\texttt{P}) = G_2$
- Aber $G \cup \mu_1(\texttt{P}) \cup \mu_2(\texttt{P})$ ist keine Konsequenz (enthält G_3)
- Problem: Wir haben unbeabsichtigt Koreferenzen eingeführt

Agenda

- Einleitung und Motivation
- Bedingungen für die Erweiterungen der Musterauswertung
- Musterauswertung mit RDFS Schlussfolgern
- Implementierungsoptionen
- Musterauswertung mit Unterstützung für Datentypen
- Musterauswertung mit OWL Schlussfolgern

Probleme mit der Naiven Idee zur Auswertung (1)

Sogar ein leerer RDF Graph hat unendlich viele axiomatische Tripel als RDFS-Konsequenz:

- $\{\} \models_{\text{RDFS}}$ `rdf:_i rdf:type rdf:Property` für alle $i \in \mathbf{N}$

Abfrage

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

$\rightsquigarrow$ Abfrage hat unendlich viele Lösungen unter RDFS Entailment

Lösung (1)

- Lösung: Bindungen kommen nur von einem endlichen Vokabular

Eine partielle Funktion μ ist eine Lösung für *Bgp(*P*)* und G unter RDFS Entailment, wenn:

1. Die Domäne von μ genau die Menge der Variablen in P ist
2. Begriffe im Wertebereich von μ in G vorkommen
3. Es ein RDF Instanz Mapping σ von leeren Knoten in P zu URIs, leeren Knoten oder RDF-Literalen gibt, so dass gilt:

   Der Graph $\mu(\sigma(\mathbb{P}))$ ist RDFS-entailed von G.

Probleme mit der Naiven Idee zur Auswertung (2)

Nur das Vokabular von G zu nehmen ist zu strikt:

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

Abfrage

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

Hat keine Lösungen (`rdf:type` $\notin$ Voc(G)).

Lösung (2)

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

Eine partielle Funktion μ ist eine Lösung für *Bgp(P)* und G unter RDFS Entailment, wenn:

1. Die Domäne von μ genau die Menge der Variablen in P ist
2. Begriffe im Wertebereich von μ in G oder $\text{Voc}^-(\text{RDFS})$ vorkommen
3. Es ein RDF Instanz Mapping σ von leeren Knoten in P zu URIs, leeren Knoten oder RDF-Literalen gibt, so dass gilt:

   Der Graph $\mu(\sigma(\mathtt{P}))$ ist RDFS-entailed von G.

Probleme mit der Naiven Idee zur Auswertung (3)

Leere Knoten haben eine existentielle Semantik

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

Probleme mit der Naiven Idee zur Auswertung (3)

Leere Knoten haben eine existentielle Semantik

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

Endliche Resultate durch Beschränkung des Wertebereichs:

Abfrage

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

Daten

G_1 = `{ ex:s1 ex:p1 _:a }`

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

Probleme mit der Naiven Idee zur Auswertung (3)

Leere Knoten haben eine existentielle Semantik

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

Endliche Resultate durch Beschränkung des Wertebereichs:

Abfrage

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

Daten

G_1 = `{ ex:s1 ex:p1 _:a }`

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

- Hat 1 Lösung für G_1 und 2 Lösungen für G_2

Probleme mit der Naiven Idee zur Auswertung (3)

Leere Knoten haben eine existentielle Semantik

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

Endliche Resultate durch Beschränkung des Wertebereichs:

Abfrage

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

Daten

G_1 = `{ ex:s1 ex:p1 _:a }`

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

- Hat 1 Lösung für G_1 und 2 Lösungen für G_2
- Hinzufügen eines Tripels, dass nichts mit dem ersten zu tun hat, führt zu neuen Lösungen

Probleme mit der Naiven Idee zur Auswertung (3)

Leere Knoten haben eine existentielle Semantik

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

Endliche Resultate durch Beschränkung des Wertebereichs:

Abfrage

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

Daten

G_1 = `{ ex:s1 ex:p1 _:a }`

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

- Hat 1 Lösung für G_1 und 2 Lösungen für G_2
- Hinzufügen eines Tripels, dass nichts mit dem ersten zu tun hat, führt zu neuen Lösungen
- Lösung: Skolemisierung

Skolemisierung

- Skolemisierung: Wir betrachten die leeren Knoten als Konstanten

Definition

Sei das Präfix `skol` eine Namensraum IRI die nicht als Präfix einer IRI im aktiven Graphen oder der Abfrage verwendet wird. Die *Skolemisierung* sk(_:`id`) eines leeren Knotens _:`id` ist definiert als sk(_:`id`) = `skol:id`. Wir erweitern sk($\cdot$) entsprechend auf Graphen.

Beispiel: Skolemisierung

Abfrage

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

Daten (Skolemisiert)

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

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

Beispiel: Skolemisierung

Abfrage

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

Daten (Skolemisiert)

$sk(G_1)$ = { 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 }

Beispiel: Skolemisierung

Abfrage

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

Daten (Skolemisiert)

$sk(G_1)$ = { 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 }

Beispiel: Skolemisierung

Abfrage

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

Daten (Skolemisiert)

$sk(G_1)$ = { 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 }
$sk(G_1) \not\models_{RDFS}$ { 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 }

Beispiel: Skolemisierung

Abfrage

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

Daten (Skolemisiert)

$sk(G_1)$ = { 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 }
$sk(G_1) \not\models_{RDFS}$ { 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 }

Beispiel: Skolemisierung

Abfrage

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

Daten (Skolemisiert)

$sk(G_1)$ = { 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 }
$sk(G_1) \not\models_{RDFS}$ { 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 }

Beispiel: Skolemisierung

Abfrage

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

Daten (Skolemisiert)

$sk(G_1)$ = { 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 }
$sk(G_1) \not\models_{RDFS}$ { 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 }

Beispiel: Skolemisierung

Abfrage

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

Daten (Skolemisiert)

$sk(G_1)$ = { 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 }
$sk(G_1) \not\models_{RDFS}$ { 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 }

Nur eine Lösung μ: ?x $\mapsto$ skol:a für $sk(G_1)$ und $sk(G_2)$

Probleme mit der Skolemisierung

- Wir wollen natürlich keine Skolem Konstanten in Lösungen zurückgeben

⇝ Wir nutzen Skolemisierung nur als eine Bedingung und wenden diese auf den Graphen und die Abfrage an

Probleme mit der Skolemisierung

- Wir wollen natürlich keine Skolem Konstanten in Lösungen zurückgeben

$\rightsquigarrow$ Wir nutzen Skolemisierung nur als eine Bedingung und wenden diese auf den Graphen und die Abfrage an

Eine partielle Funktion μ ist eine Lösung für *Bgp*($\mathtt{P}$) und G unter RDFS Entailment, wenn:

1. Die Domäne von μ genau die Menge der Variablen in $\mathtt{P}$ ist
2. Begriffe im Wertebereich von μ in G oder $\mathrm{Voc}^{-}(\mathrm{RDFS})$ vorkommen
3. Es ein RDF Instanz Mapping σ von leeren Knoten in $\mathtt{P}$ zu URIs, leeren Knoten oder RDF-Literalen gibt, so dass gilt:

   Der Graph $\mathrm{sk}(\mu(\sigma(\mathtt{P})))$ ist RDFS-entailed von $\mathrm{sk}(\mathrm{G})$.

SPARQL Lösungen unter RDFS Entailment

Definition (SPARQL Lösungen unter RDFS Entailment)

Eine partielle Funktion μ ist eine Lösung für *Bgp(*$\mathtt{P}$*)* und G unter RDFS Entailment, wenn:

1. Die Domäne von μ genau die Menge der Variablen in P ist
2. Begriffe im Wertebereich von μ in G oder $\mathrm{Voc}^{-}(\mathrm{RDFS})$ vorkommen
3. Es ein RDF Instanz Mapping σ von leeren Knoten in $\mathtt{P}$ zu URIs, leeren Knoten oder RDF-Literalen gibt, so dass gilt:

   Der Graph $\mathrm{sk}(\mu(\sigma(\mathtt{P})))$ ist wohlgeformt und RDFS-entailed von $\mathrm{sk}(G)$.

SPARQL Lösungen unter RDFS Entailment

Definition (SPARQL Lösungen unter RDFS Entailment)

Eine partielle Funktion μ ist eine Lösung für *Bgp(*$\mathtt{P}$*)* und G unter RDFS Entailment, wenn:

1. Die Domäne von μ genau die Menge der Variablen in P ist
2. Begriffe im Wertebereich von μ in G oder $\mathrm{Voc}^{-}(\mathrm{RDFS})$ vorkommen
3. Es ein RDF Instanz Mapping σ von leeren Knoten in $\mathtt{P}$ zu URIs, leeren Knoten oder RDF-Literalen gibt, so dass gilt:

   Der Graph $\mathrm{sk}(\mu(\sigma(\mathtt{P})))$ ist wohlgeformt und RDFS-entailed von $\mathrm{sk}(G)$.

Wohlgeformtheit verhindert Literale in Subjekt Position

SPARQL Entailment Regime

SPARQL Entailment Regime definieren

- Einen Name für das Regime
- Welche Entailment Relation genutzt wird ($\rightsquigarrow$ z.B. RDFS Entailment)
- Konditionen um die Bedingungen des Erweiterungspunktes für die Auswertung des Bgp($\cdot$) Operators (z.B. Endlichkeit $\rightsquigarrow$ haben wir gemacht)
- Wohlgeformte Graphen und Abfragen für das Regime ($\rightsquigarrow$ für RDFS alle RDF Graphs und SPARQL Abfragen)
- Behandlung von Inkonsistenz
- Fehlerbehandlung
- Begriffe zur Beschreibung des Regimes in Service Descriptions

Standard SPARQL Semantik als Entailment Regime

Definition (Lösungen unter einfachem Entailment)

Eine partielle Funktion μ ist eine Lösung für *Bgp*($\mathtt{P}$) und G unter ~~RDFS~~ einfachem Entailment, wenn:

1. Die Domäne von μ genau die Menge der Variablen in $\mathtt{P}$ ist
2. Begriffe im Wertebereich von μ in G ~~oder $\mathtt{Voc}^{=}(\mathtt{RDFS})$~~ vorkommen
3. Es ein RDF Instanz Mapping σ von leeren Knoten in $\mathtt{P}$ zu URIs, leeren Knoten oder RDF-Literalen gibt, so dass gilt:

   Der Graph $\mathrm{sk}(\mu(\sigma(\mathtt{P})))$ ist wohlgeformt und ~~RDFS~~ einfach entailed von $\mathrm{sk}(\mathrm{G})$.

Standard SPARQL Semantik als Entailment Regime

Definition (Lösungen unter einfachem Entailment)

Eine partielle Funktion μ ist eine Lösung für *Bgp*($\mathtt{P}$) und G unter ~~RDFS~~ einfachem Entailment, wenn:

1. Die Domäne von μ genau die Menge der Variablen in $\mathtt{P}$ ist
2. Begriffe im Wertebereich von μ in G ~~oder $\mathrm{Voc}^{=}(\mathrm{RDFS})$~~ vorkommen
3. Es ein RDF Instanz Mapping σ von leeren Knoten in $\mathtt{P}$ zu URIs, leeren Knoten oder RDF-Literalen gibt, so dass gilt:

   Der Graph $\mathrm{sk}(\mu(\sigma(\mathtt{P})))$ ist wohlgeformt und ~~RDFS~~ einfach entailed von $\mathrm{sk}(G)$.

$\rightsquigarrow$ Gleiche Definition nur mit einfachem Entailment entspricht Identifizierung von Teilgraphen

Agenda

- Einleitung und Motivation
- Bedingungen für die Erweiterungen der Musterauswertung
- Musterauswertung mit RDFS Schlussfolgern
- Implementierungsoptionen
- Musterauswertung mit Unterstützung für Datentypen
- Musterauswertung mit OWL Schlussfolgern

Implementierung des RDFS Entailment Regimes

Die Definition von Lösungen unter Entailment erlaubt verschiedene Implementierungstechniken

- Materialisierung / forwards-chaining
- Umschreiben von Abfragen / backwards-chaining
- Hybride Ansätze

Das RDFS Entailment Regime durch Materialisierung

Abfrage

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

Daten

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

- Keine Antwort unter einfachem Entailment

Das RDFS Entailment Regime durch Materialisierung

Abfrage

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

Daten

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

- Keine Antwort unter einfachem Entailment
- Idee: Wir erweitern den abgefragten Graphen mit relevanten Konsequenzen

Das RDFS Entailment Regime durch Materialisierung

Abfrage

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

Daten

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

Das RDFS Entailment Regime durch Materialisierung

Abfrage

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

Daten

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

Das RDFS Entailment Regime durch Materialisierung

Abfrage

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

Daten

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

Das RDFS Entailment Regime durch Materialisierung

Abfrage

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

Daten

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

Das RDFS Entailment Regime durch Materialisierung

Abfrage

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

Daten

```
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 .
```

Das RDFS Entailment Regime durch Materialisierung

Abfrage

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

Daten

```
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 .
```

- Abfrage über erweitertem Graph: μ: `?x` $\mapsto$ `ex:Birte`

Das RDFS Entailment Regime durch Materialisierung

Abfrage

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

Daten

```
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 .
```

- Abfrage über erweitertem Graph: μ: `?x` $\mapsto$ `ex:Birte`
- Nachteile:

Das RDFS Entailment Regime durch Materialisierung

Abfrage

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

Daten

```
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 .
```

- Abfrage über erweitertem Graph: μ: `?x` $\mapsto$ `ex:Birte`
- Nachteile:
  - Die Größe des abgefragten Graphen wächst
  - Jede Änderung erfordert eine neue Vervollständigung des Graphen

RDFS Entailment Regime durch Query Rewriting

Abfrage

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

Daten

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

- Idee: Wir erweitern die Abfrage und nicht den Graphen

RDFS Entailment Regime durch Query Rewriting

Abfrage

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

Daten

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

- Idee: Wir erweitern die Abfrage und nicht den Graphen
- Regel rdfs9 produziert eine relevante Konsequenz

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

RDFS Entailment Regime durch Query Rewriting

Abfrage

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

Daten

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

- Idee: Wir erweitern die Abfrage und nicht den Graphen
- Regel rdfs9 produziert eine relevante Konsequenz

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

RDFS Entailment Regime durch Query Rewriting

Abfrage

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

Daten

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

- Regel rdfs2 produziert nun ebenfalls eine relevante Konsequenz

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

RDFS Entailment Regime durch Query Rewriting

Abfrage

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

Daten

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

- Regel rdfs2 produziert nun ebenfalls eine relevante Konsequenz

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

RDFS Entailment Regime durch Query Rewriting

Abfrage

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

Daten

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

- Lösung μ: `?x` $\mapsto$ `ex:Birte` (vom 3. Disjunkt)

RDFS Entailment Regime durch Query Rewriting

Abfrage

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

Daten

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

- Lösung μ: `?x` $\mapsto$ `ex:Birte` (vom 3. Disjunkt)
- Nachteile:

RDFS Entailment Regime durch Query Rewriting

Abfrage

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

Daten

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

- Lösung μ: ?x $\mapsto$ ex : Birte (vom 3. Disjunkt)
- Nachteile:
  - Schwer alle Lösungen zu finden (wenn RDFS Vokabular ungewöhnlich verwendet wird)
  - Query Rewriting erfolgt pro Abfrage $\rightsquigarrow$ Laufzeit jeder Abfrage wird langsamer

Hybride Ansätze

- Kombination von Materialisierung und Query Rewriting
- Z.B. in der Praxis wird die Semantik von `owl:sameAs` nicht materialisiert
- Typisch für Individuenenafragen mittels extrahiertem Schema-Teil

Agenda

- Einleitung und Motivation
- Bedingungen für die Erweiterungen der Musterauswertung
- Musterauswertung mit RDFS Schlussfolgern
- Implementierungsoptionen
- Musterauswertung mit Unterstützung für Datentypen
- Musterauswertung mit OWL Schlussfolgern

Notiz zu Datatype Entailment

Daten

```
ex:myTable ex:hasHeight "100.5"^^xsd:decimal .
ex:myTable ex:hasHeight "+100.5"^^xsd:decimal .
```

Abfrage

```
SELECT ?h WHERE { ex:myTable ex:hasHeight ?h }
```

- Wie viele Antworten sollte die Abfrage haben?

Notiz zu Datatype Entailment

Daten

```
ex:myTable ex:hasHeight "100.5"^^xsd:decimal .
ex:myTable ex:hasHeight "+100.5"^^xsd:decimal .
```

Abfrage

```
SELECT ?h WHERE { ex:myTable ex:hasHeight ?h }
```

- Wie viele Antworten sollte die Abfrage haben?
- Viele SPARQL Implementierungen kanonikalisieren Literale $\rightsquigarrow$ Nur erstes Tripel bleibt

Notiz zu Datatype Entailment

Daten

```
ex:myTable ex:hasHeight "100.5"^^xsd:decimal .
ex:myTable ex:hasHeight "+100.5"^^xsd:decimal .
```

Abfrage

```
SELECT ?h WHERE { ex:myTable ex:hasHeight ?h }
```

- Wie viele Antworten sollte die Abfrage haben?
- Viele SPARQL Implementierungen kanonikalisieren Literale $\rightsquigarrow$ Nur erstes Tripel bleibt
- Ohne Beschränkung auf Literale die im Graphen vorkommen, hat die Anfrage unendlich viele Antworten: 100.5, 100.50, 100.500, ...
- Entailment Regimes für D-Entailment müssen laut Spezifikation kanonische Literalformen zurückgeben

Agenda

- Einleitung und Motivation
- Bedingungen für die Erweiterungen der Musterauswertung
- Musterauswertung mit RDFS Schlussfolgern
- Implementierungsoptionen
- Musterauswertung mit Unterstützung für Datentypen
- Musterauswertung mit OWL Schlussfolgern

SPARQL mit OWL's Direct Semantics

Wie können wir OWL's Direct Semantics mit SPARQL nutzen?

1. Basiert auf Beschreibungslogik (BL)
2. Semantik definiert an Hand von strukturellen OWL Objekten bzw. BL Konstruktoren
   - `owl:intersectionOf`, `ObjectIntersectionOf`, $\sqcap$
3. OWL DL Ontologien können in RDF Graphen gemapped/transformiert werden
4. Nicht jeder RDF Graph ist wohlgeformt und in einen OWL DL Ontologie umwandelbar

Mapping zwischen RDF Graphen und OWL Objekten

Turtle Syntax

ex:Harry a [a owl:Class ;
owl:unionOf (ex:GoldenEagle ex:HaastsEagle)
]. ex:HaastsEagle rdfs:subClassOf owl:Nothing .

Beschreibungslogik Syntax

$(\texttt{ex:GoldenEagle} \sqcup \texttt{ex:HaastsEagle})(\texttt{ex:Harry})$
$\texttt{ex:HaastsEagle} \sqsubseteq \bot$

OWL Functional Style Syntax

ClassAssertion(ObjectUnionOf(ex:GoldenEagle ex:HaastsEagle)
ex:Harry)
SubClassOf(ex:HaastsEagle owl:Nothing)

SPARQL mit OWL Direct Semantics

1. Das OWL Direct Semantics Entailment Regime ist beschränkt auf wohlgeformte RDF Graphen, die in OWL DL Ontologien transformiert werden können
2. Typen Deklarationen erforderlich zur Disambiguierung
   - `ex:subProp rdfs:subPropertyOf ex:superProp .`

SPARQL mit OWL Direct Semantics

1. Das OWL Direct Semantics Entailment Regime ist beschränkt auf wohlgeformte RDF Graphen, die in OWL DL Ontologien transformiert werden können
2. Typen Deklarationen erforderlich zur Disambiguierung
   - `ex:subProp rdfs:subPropertyOf ex:superProp .`
   - `ex:subProp a owl:ObjectProperty .`
   - `ex:superProp a owl:ObjectProperty .`

SPARQL mit OWL Direct Semantics

1. Das OWL Direct Semantics Entailment Regime ist beschränkt auf wohlgeformte RDF Graphen, die in OWL DL Ontologien transformiert werden können
2. Typen Deklarationen erforderlich zur Disambiguierung
   - `ex:subProp rdfs:subPropertyOf ex:superProp .`
   - `ex:subProp a owl:ObjectProperty .`
   - `ex:superProp a owl:ObjectProperty .`
3. Muster werden in erweiterte OWL Objekte mit Variablen transformiert

SPARQL mit OWL Direct Semantics

1. Das OWL Direct Semantics Entailment Regime ist beschränkt auf wohlgeformte RDF Graphen, die in OWL DL Ontologien transformiert werden können
2. Typen Deklarationen erforderlich zur Disambiguierung
   - `ex:subProp rdfs:subPropertyOf ex:superProp .`
   - `ex:subProp a owl:ObjectProperty .`
   - `ex:superProp a owl:ObjectProperty .`
3. Muster werden in erweiterte OWL Objekte mit Variablen transformiert
4. Variablen dürfen in der Position von Klassen, Propertys, Individuen, oder Literalen vorkommen
   - Variables brauchen teilweise auch Typen Deklarationen
   - `?x rdfs:subPropertyOf ?y .`

Berechnung einfacher Graphmuster unter OWL DS

Definition (Lösungen unter OWL's Direct Semantics)

Eine partielle Funktion μ ist eine Lösung für *Bgp(P)* und G unter OWL Direct Semantics (OWL DS) Entailment, wenn:

1. G und P in eine OWL DL Ontologie $O(G)$ bzw. OWL DL Axiome $O(P)$ transformiert werden können
2. Die Domäne von μ genau die Menge der Variablen in P ist
3. Begriffe im Wertebereich von μ in G oder $\text{Voc}^-(\text{OWL})$ vorkommen
4. Es ein RDF Instanz Mapping σ von *anonymen Individuen* (leeren Knoten) in $O(P)$ zu URIs, leeren Knoten oder RDF-Literalen gibt, so dass gilt:

   $O(G) \cup \mu(\sigma(O(P)))$ ist eine OWL DL Ontologie und die Axiome $\mu(\sigma(O(P)))$ sind OWL DS-entailed von $O(G)$

SPARQL OWL Direct Semantics Beispiel

Abfrage mit Muster P

```
SELECT ?type WHERE { ex:Harry a ?type }
```

Graph G

```
_:ont a owl:Ontology .
ex:Harry a [ a owl:Class ; owl:unionOf
           (ex:GoldenEagle ex:HaastsEagle) ] .
ex:HaastsEagle rdfs:subClassOf owl:Nothing .
```

SPARQL OWL Direct Semantics Beispiel

Abfrage mit Muster P und O(P)

```
SELECT ?type WHERE { ex:Harry a ?type }
SELECT ?type WHERE { ?type(ex:Harry) }
```

Graph G und O(G)

```
_:ont a owl:Ontology .
ex:Harry a [ a owl:Class ; owl:unionOf
           (ex:GoldenEagle ex:HaastsEagle) ] .
ex:HaastsEagle rdfs:subClassOf owl:Nothing .

(ex:GoldenEagle ⊔ ex:HaastsEagle)(ex:Harry)
ex:HaastsEagle ⊑ owl:Nothing
```

Beispiel in Funktional Style Syntax

Abfrage mit Muster P und O(P)

```
SELECT ?type WHERE { ex:Harry a ?type }
SELECT ?type WHERE {ClassAssertion(?type ex:Harry)}
```

Graph G und O(G)

```
_:ont a owl:Ontology .
ex:Harry a [ a owl:Class ; owl:unionOf
           (ex:GoldenEagle ex:HaastsEagle) ] .
ex:HaastsEagle rdfs:subClassOf owl:Nothing .

Ontology(
  ClassAssertion(ObjectUnionOf(
    ex:GoldenEagle ex:HaastsEagle) ex:Harry)
SubClassOf(ex:HaastsEagle owl:Nothing)
```

SPARQL OWL DS Beispiel

Muster O(P) und O(G)

$?type(ex:Harry)$

$(ex:GoldenEagle \sqcup ex:HaastsEagle)(ex:Harry)$
$ex:HaastsEagle \sqsubseteq owl:Nothing$

- Nur Bindungen mit dem entsprechenden Variablentyp (hier Klassennamen) müssen getestet werden
  ($O(G) \cup O(\mu(\sigma(P))$ muss eine OWL DL Ontologie sein)
- Reasoning Tools können Entailment testen:
  - O(G) $\models_{\text{OWL DS}} ex:GoldenEagle(ex:Harry)$
  - O(G) $\models_{\text{OWL DS}} ex:HaastsEagle(ex:Harry)$
- Oder wir fragen den Reasoner nach Typen von ex:Harry

Ergebnis: Eine Lösung μ: $?type \mapsto ex:GoldenEagle$

Implementierung des OWL DS Regimes

- Materialisierung nicht möglich

Implementierung des OWL DS Regimes

- Materialisierung nicht möglich

⇝ Z.B. können wir beliebige Disjunktionen in der Abfrage haben:

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

Implementierung des OWL DS Regimes

- Materialisierung nicht möglich

$\rightsquigarrow$ Z.B. können wir beliebige Disjunktionen in der Abfrage haben:

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

- Turtle ist keine besonders passende Syntax für OWL

  $\rightsquigarrow$ Usability Probleme

Implementierung des OWL DS Regimes

- Abfragen gehen über einfache Instanzanfragen hinaus
- Optimierung sehr schwer für komplexe Abfragen

```
SELECT ?c WHERE { ex:Birte a [ a owl:Restriction
; owl:onProperty ex:givesLecture ;
owl:someValuesFrom ?c ] }

SELECT ?c WHERE { ClassAssertion(
ObjectSomeValuesFrom( ex:givesLecture ?c )
ex:Birte ) }

SELECT ?c WHERE {
(∃ex : givesLecture.?c)(ex : Birte) }
```

- ⇝ Oft müssen alle möglichen Bindungen getestet werden

SPARQL mit OWL Profilen

Die OWL Profile sind besser geeignet für Web Anwendungen

- Das OWL RL Profil kann über Materialisierung implementiert werden
- Polynomielle Komplexität
- Erweitert die RDF(S) Semantik (also kann auch mit OWL's RDF-Based Semantics verwendet werden)
- Arbeitet mit beliebigen RDF Graphen

Weitere Entailment Regimes

- RDF Entailment Regime (Vereinfachung des RDFS Regimes)
- D-Entailment Regime (Erweiterung des RDFS Regimes mit Unterstützung für Datentypen)
- RIF Core Entailment Regime (Regeln plus Tripel Daten)
  - Abfrage wird bzgl. eines RDF Graphs und einer Menge von Regeln beantwortet

Weitere Entailment Regimes

Daten

```
ex:Adrian ex:childOf ex:Uwe .
ex:Adrian rdf:type ex:Male .
ex:Uwe rdf:type ex:Male .
<myRules.rifps> rif:usedWithProfile ent:Simple .
```

Regel(n)

```
Group ( Forall ?X ?Y (
  ?Y [ ex:fatherOf -> ?X ] :- And( ?X [ ex:childOf -> ?Y  ]
                                   ?Y [ rdf:type -> ex:Male ] )
 ) )
```

Abfrage

```
SELECT ?father ?child WHERE {?father ex:fatherOf ?child }
```

$\rightsquigarrow$ Lösung $\texttt{?father} \mapsto \texttt{ex:Uwe}, \texttt{?child} \mapsto \texttt{ex:Adrian}$

Zusammenfassung

- SPARQL kann nun auch für RDF(S), OWL, und RIF verwendet werden
- Entailment Regime werden zur Zeit vom W3C standardisiert
- Überschreiben die Auswertung des $\mathrm{Bgp}(\cdot)$ Operators
- Property Paths aus SPARQL Query 1.1 problematisch
- Definition von Lösungen (relativ) allgemeingültig
  - Passt auch für einfaches Entailment
  - OWL's Direct Semantics braucht einige extra Bedingungen
- Implementierung und Effizienz für OWL schwierig
  $\rightsquigarrow$ Profile