ulm university universität ulm

Semantic Web Grundlagen

Semantik von SPARQL

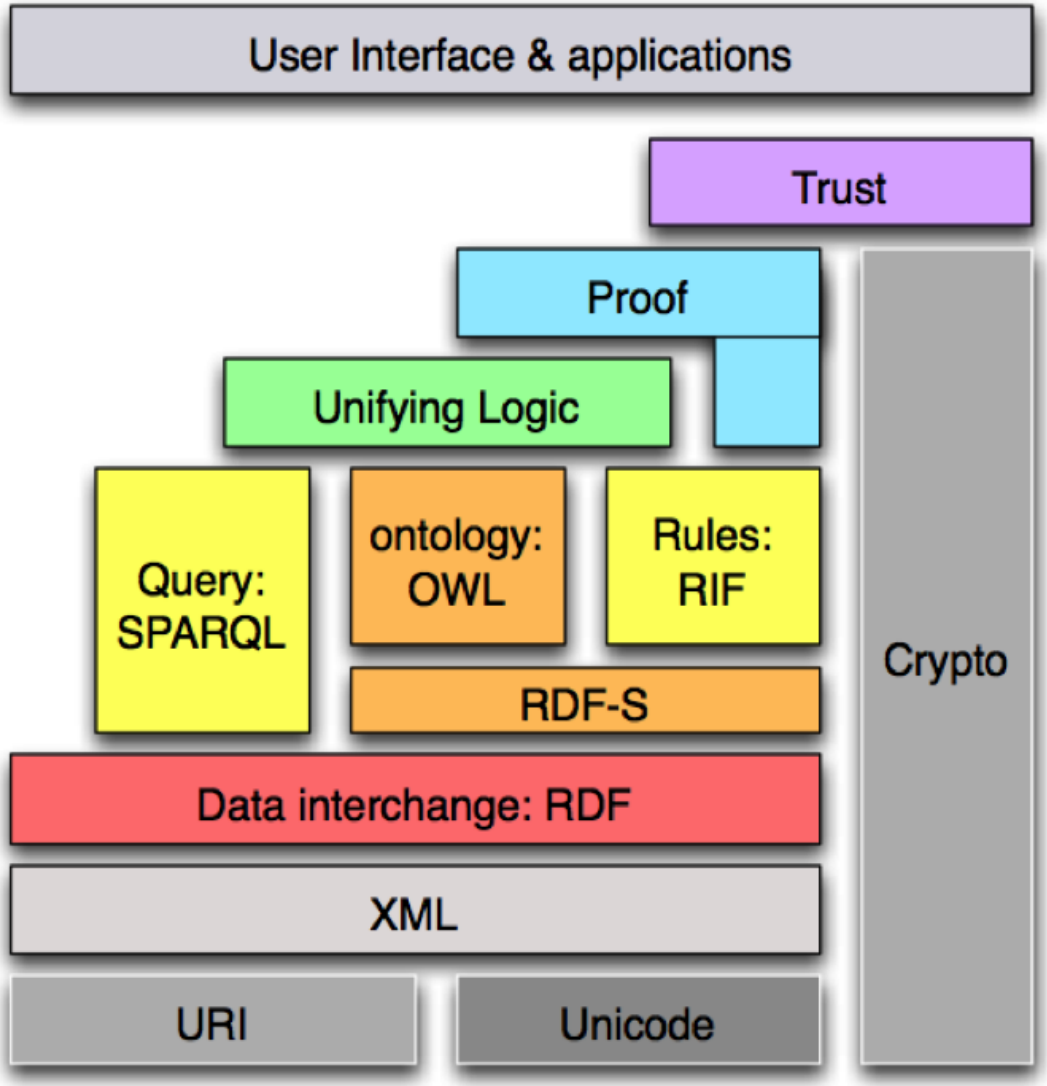
Birte Glimm
Institut für Künstliche Intelligenz | 22. Dez 2011

Organisatorisches: Inhalt

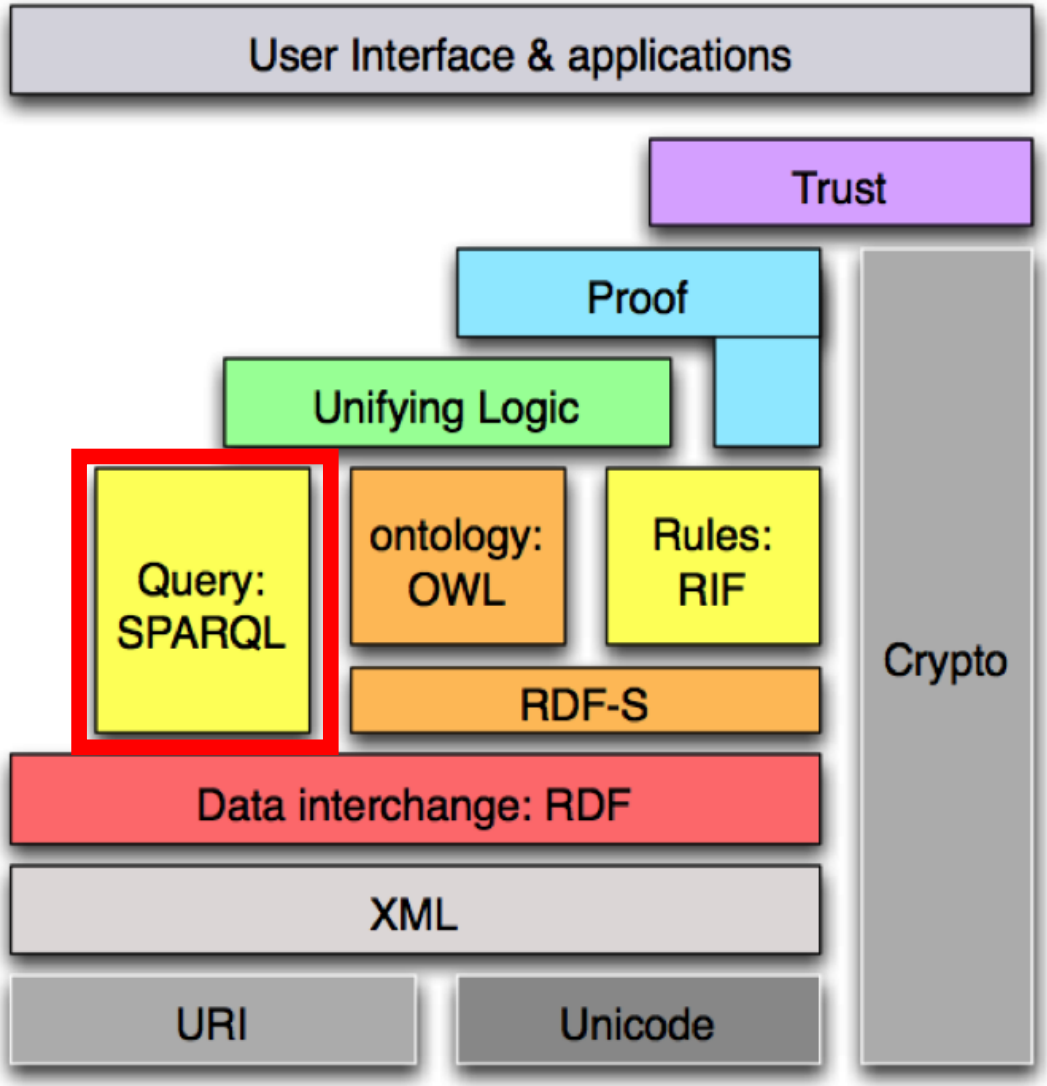
Einleitung und XML	17. Okt	Hypertableau II	12. Dez
Einführung in RDF	20. Okt	Übung 4	15. Dez
RDF Schema	24. Okt	SPARQL Syntax & Intuition	19. Dez
fällt aus	27. Okt	SPARQL Semantik	22. Dez
Logik – Grundlagen	31. Okt	SPARQL 1.1	9. Jan
Übung 1	3. Nov	Übung 5	12. Jan
Semantik von RDF(S)	7. Nov	SPARQL Entailment	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

- Wiederholung
- Auswertung der SPARQL-Algebra
- SPARQL Algebra Transformation
- Operationen für die Modifikatoren
- Abstrakte SPARQL Abfragen
- Zusammenfassung

Agenda

- Wiederholung
- Auswertung der SPARQL-Algebra
- SPARQL Algebra Transformation
- Operationen für die Modifikatoren
- Abstrakte SPARQL Abfragen
- Zusammenfassung

Recap: Vorgestellte SPARQL-Merkmale

Grundstruktur
PREFIX
WHERE

Ausgabeformate
SELECT
CONSTRUCT
ASK
DESCRIBE

Graph-Muster
Basic Graph Patterns
{...}
OPTIONAL
UNION
GRAPH

Filter
BOUND
isURI
isBLANK
isLITERAL
STR
LANG
DATATYPE
sameTERM
langMATCHES
REGEX

Modifikatoren
ORDER BY
LIMIT
OFFSET
DISTINCT

Übersetzung in SPARQL-Algebra

```
{ ?book ex:Price ?price .
  FILTER (?price < 15)
  OPTIONAL { ?book ex:Titel ?titel }
  { ?book ex:Author ex:Shakespeare } UNION
  { ?book ex:Author ex:Marlowe }
}
```

Semantik einer SPARQL-Anfrage:

1. Umwandlung der Anfrage in einen algebraischen Ausdruck
2. Berechnung des Ergebnisses dieses Ausdrucks

Übersetzung in SPARQL-Algebra

```
{ ?book ex:Price ?price
  FILTER (?price < 15)
  OPTIONAL { ?book ex:Titel ?titel }
  { ?book ex:Author ex:Shakespeare } UNION
  { ?book ex:Author ex:Marlowe }
}
```

Achtung: Filter beziehen sich immer auf die gesamte Gruppe in der sie auftauchen

Übersetzung in SPARQL-Algebra

```
{ ?book ex:Price ?price
  OPTIONAL { ?book ex:Titel ?titel }
  { ?book ex:Author ex:Shakespeare } UNION
  { ?book ex:Author ex:Marlowe }
  FILTER (?price < 15)
}
```

1. Auflösen von abgekürzten IRIs

Übersetzung in SPARQL-Algebra

```
{ ?book <http://ex.org/Price> ?price
  OPTIONAL { ?book <http://ex.org/Titel> ?titel }
  { ?book <http://eg.org/Author>
                  <http://eg.org/Shakespeare> } UNION
  { ?book <http://eg.org/Author>
                  <http://eg.org/Marlowe> }
  FILTER (?price < 15)
}
```

Übersetzung in SPARQL-Algebra

```
{ ?book <http://ex.org/Price> ?price
  OPTIONAL { ?book <http://ex.org/Titel> ?titel }
  { ?book <http://eg.org/Author>
                  <http://eg.org/Shakespeare> } UNION
  { ?book <http://eg.org/Author>
                  <http://eg.org/Marlowe> }
  FILTER (?price < 15)
}
```

2. Ersetzung einfacher Graph-Muster mit Operator Bgp(·)

Übersetzung in SPARQL-Algebra

```
{ Bgp(?book <http://eg.org/Price> ?price)
  OPTIONAL {Bgp(?book <http://eg.org/Titel> ?titel)}
  {Bgp(?book <http://eg.org/Author>
                    <http://eg.org/Shakespeare>)} UNION
  {Bgp(?book <http://eg.org/Author>
                    <http://eg.org/Marlowe>)}
  FILTER (?price < 15)
}
```

Übersetzung in SPARQL-Algebra

```
{ Bgp(?book <http://eg.org/Price> ?price)
  OPTIONAL {Bgp(?book <http://eg.org/Titel> ?titel)}
  {Bgp(?book <http://eg.org/Author>
                    <http://eg.org/Shakespeare>)} UNION
  {Bgp(?book <http://eg.org/Author>
                    <http://eg.org/Marlowe>)}
  FILTER (?price < 15)
}
```

3. Einfügung des LeftJoin(·) Operators für optionale Teile

Übersetzung in SPARQL-Algebra

```
{ LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
           Bgp(?book <http://eg.org/Titel> ?titel),
           true)
  {Bgp(?book <http://eg.org/Author>
                <http://eg.org/Shakespeare>)} UNION
  {Bgp(?book <http://eg.org/Author>
                <http://eg.org/Marlowe>)}
  FILTER (?price < 15)
}
```

Übersetzung in SPARQL-Algebra

```
{ LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
           Bgp(?book <http://eg.org/Titel> ?titel),
           true)
  {Bgp(?book <http://eg.org/Author>
             <http://eg.org/Shakespeare>)} UNION
  {Bgp(?book <http://eg.org/Author>
             <http://eg.org/Marlowe>)}
  FILTER (?price < 15)
}
```

4. Kombination alternativer Muster mit Union($\cdot$) Operator

$\rightsquigarrow$ Bezieht sich auf benachbarte Muster und hat höhere Priorität als Konjunktion (linksassoziativ)

Übersetzung in SPARQL-Algebra

```
{ LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
           Bgp(?book <http://eg.org/Titel> ?titel),
           true)
  Union(Bgp(?book <http://eg.org/Author>
                  <http://eg.org/Shakespeare>),
        Bgp(?book <http://eg.org/Author>
                  <http://eg.org/Marlowe>))
  FILTER (?price < 15)
}
```

Übersetzung in SPARQL-Algebra

```
{ LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
           Bgp(?book <http://eg.org/Titel> ?titel),
           true)
  Union(Bgp(?book <http://eg.org/Author>
                  <http://eg.org/Shakespeare>),
        Bgp(?book <http://eg.org/Author>
                  <http://eg.org/Marlowe>))
  FILTER (?price < 15)
}
```

5. Anwendung des Join(·) Operators um nicht-Filter Elemente zu verbinden

Übersetzung in SPARQL-Algebra

```
{ Join(
    LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
             Bgp(?book <http://eg.org/Titel> ?titel),
             true),
    Union(Bgp(?book <http://eg.org/Author>
                    <http://eg.org/Shakespeare>),
          Bgp(?book <http://eg.org/Author>
                    <http://eg.org/Marlowe>)))
    FILTER (?price < 15)
}
```

Übersetzung in SPARQL-Algebra

```
{ Join(
    LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
             Bgp(?book <http://eg.org/Titel> ?titel),
             true),
    Union(Bgp(?book <http://eg.org/Author>
                    <http://eg.org/Shakespeare>),
          Bgp(?book <http://eg.org/Author>
                    <http://eg.org/Marlowe>)))
    FILTER (?price < 15)
}
```

6. Übersetzung einer Gruppe mit Filtern durch Filter(·) Operator

Übersetzung in SPARQL-Algebra

```
Filter(?price < 15,
    Join(
      LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
               Bgp(?book <http://eg.org/Titel> ?titel),
               true),
      Union(Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Shakespeare>),
            Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Marlowe>))))
```

Übersetzung in SPARQL-Algebra

```
Filter(?price < 15,
    Join(
      LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
               Bgp(?book <http://eg.org/Titel> ?titel),
               true),
      Union(Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Shakespeare>),
            Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Marlowe>))))
```

- Online Umwandlungstool:
  http://sparql.org/query-validator.html

Agenda

- Wiederholung
- Auswertung der SPARQL-Algebra
- SPARQL Algebra Transformation
- Operationen für die Modifikatoren
- Abstrakte SPARQL Abfragen
- Zusammenfassung

Bedeutung der SPARQL Algebra Operationen

Nun haben wir ein Algebra Objekt, aber was bedeuten die Operatoren?

Bgp(P)	Auswertung des Musters P
Join(A_1, A_2)	Konjunktive Verknüpfung der Ergebnisse von A_1 und A_2
Union(A_1, A_2)	Vereinigung der Ergebnisse von A_1 und A_2
LeftJoin(A_1, A_2, F)	Optionale Verknüpfung der Ergebnisse von A_1 und A_2 mit Filterbedingung F (`true` wenn kein Filter gegeben)
Filter(F, A)	Filterung der Ergebnisse von A auf Bedingung F
Z	Konstante füre *leeren Ausdruck*
Graph(varOrIRI, A)	Auswertung von A über dem Graphen IRI bzw. über allen benannten Graphen

Definition der SPARQL Operatoren

Wie können wir das formal definieren?

Ausgabe:

- “Ergebnistabelle” (solution set, Formatierung nicht relevant)

Eingabe:

- Angefragte RDF Datenbasis (active graph)
- Teilergebnisse von Unterausdrücken
- Verschiedene Parameter je nach Operation

$\rightsquigarrow$ Wie sollen “Ergebnisse” formal dargestellt werden?

SPARQL-Ergebnisse

Intuition: Ergebnisse kodieren Tabellen mit Variablenbelegungen

Ergebnis:

Liste von *Lösungen* (Lösungssequenz)

$\rightsquigarrow$ jede Lösung entspricht einer Tabellenzeile

SPARQL-Ergebnisse

Intuition: Ergebnisse kodieren Tabellen mit Variablenbelegungen

Ergebnis:

Liste von *Lösungen* (Lösungssequenz)

$\rightsquigarrow$ jede Lösung entspricht einer Tabellenzeile

Lösung:

Partielle Abbildung (Funktion)

- Definitionsbereich (Domäne): ausgewählte Menge von Variablen
- Wertebereich: URIs $\cup$ leere Knoten $\cup$ RDF-Literale

$\rightsquigarrow$ Ungebundene Variablen sind solche, die von einer Lösung keinen Wert zugewiesen bekommen (*partielle* Funktion)

Berechnung einfacher Graphmuster

Eine partielle Funktion μ ist eine Lösung des Ausdrucks *Bgp(P)* und des angefragten Graphen G (P: BGP), falls gilt:

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

   Der Graph $\mu(\sigma(P))$ ist ein Teilgraph des angefragten Graphen

Ergebnis der Auswertung von *Bgp(P)* über G, geschrieben $[\![Bgp(P)]\!]_G$:

Multimenge (multi set, Duplikate, Reihenfolge undefiniert) solcher Lösungen μ

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

Multimengen

Definition Multimenge

Eine Multimenge über eine Menge S ist eine totale Funktion $M\colon S \to \mathbf{N}^+ \cup \{\omega\}$

- $\mathbf{N}^+$ die positiven natürlichen Zahlen
- $\omega > n$ für alle $n \in \mathbf{N}^+$
- $M(s)$ ist die Vielfachheit von $s \in S$
- ω: zählbar unendliche Anzahl von Vorkommnissen

- Wir repräsentieren eine Multimenge über der Menge S auch mit der Menge $\{(s, M(s)) \mid s \in S\}$
- Wir schreiben $(s, n) \in M$ wenn $M(s) = n$
- Wir nehmen an, dass $M(s) = 0$ wenn $s \notin S$
- Andere Notation: $\dot{\{}a, b, b\dot{\}}$ entspricht der Multimenge M über der Menge $\{a, b\}$ mit $M(a) = 1$ und $M(b) = 2$

Beispiel Lösungsmenge

```
ex:Birte ex:gives [
    a ex:Lecture ;
    ex:hasTopic "SPARQL" ] .
ex:Sebastian ex:gives [
    a ex:Lecture ;
    ex:hasTopic "DLs and OWL" ] .
```

Bgp(`?who ex:gives _:x . _:x ex:hasTopic ?what`)

Beispiel Lösungsmenge

```
ex:Birte ex:gives _:a .
_:a rdf:type ex:Lecture .
_:a ex:hasTopic "SPARQL" .
ex:Sebastian ex:gives _:b .
_:b rdf:type ex:Lecture .
_:b ex:hasTopic "DLs and OWL" .
```

Bgp(`?who ex:gives _:x . _:x ex:hasTopic ?what`)

Beispiel Lösungsmenge

```
ex:Birte ex:gives _:a .
_:a rdf:type ex:Lecture .
_:a ex:hasTopic "SPARQL" .
ex:Sebastian ex:gives _:b .
_:b rdf:type ex:Lecture .
_:b ex:hasTopic "DLs and OWL" .
```

Bgp(`?who ex:gives _:x . _:x ex:hasTopic ?what`)

μ_1: `?who` $\mapsto$ `ex:Birte`, `?what` $\mapsto$ `"SPARQL"`

σ_1: `_:x` $\mapsto$ `_:a`

Beispiel Lösungsmenge

```
ex:Birte ex:gives _:a .
_:a rdf:type ex:Lecture .
_:a ex:hasTopic "SPARQL" .
ex:Sebastian ex:gives _:b .
_:b rdf:type ex:Lecture .
_:b ex:hasTopic "DLs and OWL" .
```

Bgp(`?who ex:gives _:x . _:x ex:hasTopic ?what`)

μ_1: `?who` $\mapsto$ `ex:Birte`, `?what` $\mapsto$ `"SPARQL"`

σ_1: `_:x` $\mapsto$ `_:a`

μ_2: `?who` $\mapsto$ `ex:Sebastian`, `?what` $\mapsto$ `"DLs and OWL"`

σ_2: `_:x` $\mapsto$ `_:b`

Beispiel Lösungsmenge

```
ex:Birte ex:gives _:a .
_:a rdf:type ex:Lecture .
_:a ex:hasTopic "SPARQL" .
ex:Sebastian ex:gives _:b .
_:b rdf:type ex:Lecture .
_:b ex:hasTopic "DLs and OWL" .
```

Bgp(`?who ex:gives _:x . _:x ex:hasTopic ?what`)

μ_1: `?who` $\mapsto$ `ex:Birte`, `?what` $\mapsto$ `"SPARQL"`

σ_1: `_:x` $\mapsto$ `_:a`

μ_2: `?who` $\mapsto$ `ex:Sebastian`, `?what` $\mapsto$ `"DLs and OWL"`

σ_2: `_:x` $\mapsto$ `_:b`

Zwei Lösungen jeweils mit Kardinalität 1

Vereinigung von Lösungen

Zwei Lösugen μ_1 und μ_2 sind *kompatibel* wenn
$\mu_1(x) = \mu_2(x)$ für alle x für die μ_1 und μ_2 definiert sind

Vereinigung von Lösungen

Zwei Lösugen μ_1 und μ_2 sind *kompatibel* wenn
$\mu_1(x) = \mu_2(x)$ für alle x für die μ_1 und μ_2 definiert sind

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b

μ_2 : ?y $\mapsto$ ex : b, ?z $\mapsto$ ex : c

Vereinigung von Lösungen

Zwei Lösugen μ_1 und μ_2 sind *kompatibel* wenn
$\mu_1(x) = \mu_2(x)$ für alle x für die μ_1 und μ_2 definiert sind

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?y $\mapsto$ ex : b, ?z $\mapsto$ ex : c ✓

Vereinigung von Lösungen

Zwei Lösugen μ_1 und μ_2 sind *kompatibel* wenn
$\mu_1(x) = \mu_2(x)$ für alle x für die μ_1 und μ_2 definiert sind

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?y $\mapsto$ ex : b, ?z $\mapsto$ ex : c ✓

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?x $\mapsto$ ex : b, ?z $\mapsto$ ex : c

Vereinigung von Lösungen

Zwei Lösugen μ_1 und μ_2 sind *kompatibel* wenn
$\mu_1(x) = \mu_2(x)$ für alle x für die μ_1 und μ_2 definiert sind

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?y $\mapsto$ ex : b, ?z $\mapsto$ ex : c ✓

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?x $\mapsto$ ex : b, ?z $\mapsto$ ex : c ↯

Vereinigung von Lösungen

Zwei Lösugen μ_1 und μ_2 sind *kompatibel* wenn
$\mu_1(x) = \mu_2(x)$ für alle x für die μ_1 und μ_2 definiert sind

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?y $\mapsto$ ex : b, ?z $\mapsto$ ex : c ✓

μ_1 : ?x $\mapsto$ ex : a
μ_2 : ?y $\mapsto$ ex : b

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?x $\mapsto$ ex : b, ?z $\mapsto$ ex : c ↯

Vereinigung von Lösungen

Zwei Lösugen μ_1 und μ_2 sind *kompatibel* wenn
$\mu_1(x) = \mu_2(x)$ für alle x für die μ_1 und μ_2 definiert sind

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?y $\mapsto$ ex : b, ?z $\mapsto$ ex : c ✓

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?x $\mapsto$ ex : b, ?z $\mapsto$ ex : c ↯

μ_1 : ?x $\mapsto$ ex : a
μ_2 : ?y $\mapsto$ ex : b ✓

Vereinigung von Lösungen

Zwei Lösugen μ_1 und μ_2 sind *kompatibel* wenn
$\mu_1(x) = \mu_2(x)$ für alle x für die μ_1 und μ_2 definiert sind

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?y $\mapsto$ ex : b, ?z $\mapsto$ ex : c ✓

μ_1 : ?x $\mapsto$ ex : a, ?y $\mapsto$ ex : b
μ_2 : ?x $\mapsto$ ex : b, ?z $\mapsto$ ex : c ↯

μ_1 : ?x $\mapsto$ ex : a
μ_2 : ?y $\mapsto$ ex : b ✓

Vereinigung von zwei kompatiblen Lösungen μ_1 und μ_2:

$$(\mu_1 \cup \mu_2)(x) = \begin{cases} \mu_1(x) & \text{wenn } x \in \text{dom}(\mu_1) \\ \mu_2(x) & \text{andernfalls} \end{cases}$$

$\rightsquigarrow$ Einfache Intuition: Vereinigung von Tabellenzeilen

Auswertung von *Filter*$(\cdot)$

- Wir können nun die Auswertung der wesentlichen Algebra Operationen definieren

Die Auswertung von *Filter*(F, A) über einem Graphen G, geschrieben $[\![\textit{Filter}(F, A)]\!]_{\mathsf{G}}$, mit F einer Filterbedingung und A einem Algebra Objekt ergibt:

$$\Big\{(\mu, n) \mid M = [\![A]\!]_{\mathsf{G}}, M(\mu) = n > 0 \text{ and } [\![\mu(F)]\!] = \text{true}\Big\}$$

$[\![\mu(F)]\!]$ ist das Boolsche Ergebnis der Filterauswertung

Auswertung von *Join*(·)

Für die Auswertung von *Join*(A_1, A_2) über einem Graphen G mit A_1, A_2 Algebra Objekten definieren wir:

- Sei $M_1 = [\![A_1]\!]_G$
- Sei $M_2 = [\![A_2]\!]_G$
- Sei $J(\mu) = \{(\mu_1, \mu_2) \mid M_1(\mu_1) > 0, M_2(\mu_2) > 0,$ μ_1 und μ_2 sind kompatibel und $\mu = \mu_1 \cup \mu_2\}$

Die Auswertung $[\![\textit{Join}(A_1, A_2)]\!]_G$ ergibt das Ergebnis

$$\left\{(\mu, n) \mid n = \sum_{(\mu_1,\mu_2)\in J(\mu)} \big(M_1(\mu_1) * M_2(\mu_2)\big) > 0\right\}$$

Beispiel zu *Join*(·)

Wir betrachten *Join*(A_1, A_2) über einem Graphen G mit $[\![A_1]\!]_G = M_1$ und $[\![A_2]\!]_G = M_2$ und:

$$
\begin{aligned}
M_1 =& \{((\mu_1 : \texttt{?x} \mapsto \texttt{ex:a}, \texttt{?y} \mapsto \texttt{ex:b}), 2), \\
& ((\mu_2 : \texttt{?x} \mapsto \texttt{ex:a}, 1)\} \\
M_2 =& \{((\mu_3 : \texttt{?y} \mapsto \texttt{ex:b}, \texttt{?z} \mapsto \texttt{ex:c}, 3)\} \\
\mu =& \texttt{?x} \mapsto \texttt{ex:a}, \texttt{?y} \mapsto \texttt{ex:b}, \texttt{?z} \mapsto \texttt{ex:c} \\
J(\mu) =& \{(\mu_1, \mu_3), (\mu_2, \mu_3)\} \\
\textit{Join}(M_1, M_2) =& \left\{ (\mu, n) \mid n = \sum_{(\mu_1, \mu_2) \in J(\mu)} \left(M_1(\mu_1) * M_2(\mu_2)\right) > 0 \right\} \\
=& \{(\mu, 9)\} \\
n =& 2 * 3 + 1 * 3 = 6 + 3 = 9
\end{aligned}
$$

Auswertung von *Union*(·)

Die Auswertung von *Union*(A_1, A_2) über einem Graphen G, geschrieben $[\![\textit{Union}(A_1, A_2)]\!]_{\mathsf{G}}$, mit A_1, A_2 Algebra Objekten ergibt:

$$\left\{(\mu, n) \mid M_1 = [\![A_1]\!]_{\mathsf{G}}, M_2 = [\![A_2]\!]_{\mathsf{G}}, n = M_1(\mu) + M_2(\mu) > 0\right\}$$

Auswertung von *LeftJoin*$(\cdot)$

Die Auswertung von *LeftJoin*(A_1, A_2, F) über einem Graphen G mit F einer Filterbedingung und A_1, A_2 Algebra Objekten definieren wir

- $M_1 = [\![A_1]\!]_G$
- $M_2 = [\![A_2]\!]_G$

Die Auswertung $[\![\textit{LeftJoin}(A_1, A_2, F)]\!]_G$ von *LeftJoin*(A_1, A_2, F) über G ergibt

$$[\![\textit{Filter}(F, \textit{Join}(A_1, A_2))]\!]_G \cup$$
$$\Big\{(\mu_1, M_1(\mu_1)) \mid \text{für alle } \mu_2 \text{ mit } M_2(\mu_2) > 0 : \mu_1 \text{ und } \mu_2 \text{ sind inkompatibel oder } [\![(\mu_1 \cup \mu_2)(F)]\!] = \texttt{false}\Big\}$$

Auswertung von Graph$(\cdot)$

Die Auswertung von Graph(varOrIRI, A) über einem Graphen G, geschrieben $[\![\text{Graph(varOrIRI, A)}]\!]_{\text{G}}$, mit A einem Algebra Objekt:

- Ist varOrIRI=I, für eine IRI I und enthält das Dataset für I einen Graphen D_I, dann ist das Ergebnis $[\![\ A\ ]\!]^{\text{D}_\text{I}}_{\text{G}}$
- Ist varOrIRI=I, für eine IRI I und enthält das Dataset für I keinen Graphen D_I, dann ist das Ergebnis die leere Menge
- Ist varOrIRI=v, für eine Variable v, dann ist das Ergebnis die Vereinigung der Lösungen über alle benannten Graphen des Datasets, wobei v jeweils an die IRI des Graphen gebunden wird

Beispiel

```
@prefix ex:  <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet          ex:Author  ex:Shakespeare ;
                   ex:Price   "10.50"^^xsd:decimal .
ex:Macbeth         ex:Author  ex:Shakespeare .
ex:Tamburlaine     ex:Author  ex:Marlowe ;
                   ex:Price   "17"^^xsd:integer .
ex:DoctorFaustus   ex:Author  ex:Marlowe ;
                   ex:Price   "12"^^xsd:integer ;
      ex:Titel  "The Tragical History of Doctor Faustus" .
ex:RomeoJulia      ex:Author  ex:Brooke ;
                   ex:Price   "9"^^xsd:integer .
```

```
{ ?book   ex:Price   ?price .  FILTER (?price < 15)
  OPTIONAL { ?book   ex:Titel   ?titel . }
  { ?book   ex:Author   ex:Shakespeare . } UNION
  { ?book   ex:Author   ex:Marlowe . }
}
```

```
@prefix ex:  <http://example.org/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
ex:Hamlet          ex:Author  ex:Shakespeare ;
                   ex:Price  "10.50"^^xsd:decimal .
ex:Macbeth         ex:Author  ex:Shakespeare .
ex:Tamburlaine     ex:Author  ex:Marlowe ;
                   ex:Price  "17"^^xsd:integer .
ex:DoctorFaustus   ex:Author  ex:Marlowe ;
                   ex:Price  "12"^^xsd:integer ;
      ex:Titel  "The Tragical History of Doctor Faustus" .
ex:RomeoJulia      ex:Author  ex:Brooke ;
                   ex:Price  "9"^^xsd:integer .
```

Filter(`?price < 15`*,*
Join(LeftJoin(Bgp(`?book <http://eg.org/Price> ?price`*),*
Bgp(`?book <http://eg.org/Titel> ?titel`*), true),*
Union(Bgp(`?book <http://eg.org/Author>`
`<http://eg.org/Shakespeare>`*),*
Bgp(`?book <http://eg.org/Author>`
`<http://eg.org/Marlowe>`*))))*

Beispielauswertung (1)

```
Filter( ?price < 15,
    Join(
      LeftJoin(Bgp( ?book <http://eg.org/Price> ?price),
               Bgp( ?book <http://eg.org/Titel> ?titel),
               true),
      Union(Bgp( ?book <http://eg.org/Author>
                       <http://eg.org/Shakespeare>),
            Bgp( ?book <http://eg.org/Author>
                       <http://eg.org/Marlowe>))))
```

book
ex:Tamburlaine
ex:DoctorFaustus

Beispielauswertung (1)

```
Filter( ?price < 15,
    Join(
      LeftJoin(Bgp( ?book <http://eg.org/Price> ?price),
               Bgp( ?book <http://eg.org/Titel> ?titel),
               true),
      Union(Bgp( ?book <http://eg.org/Author>
                       <http://eg.org/Shakespeare>),
            Bgp( ?book <http://eg.org/Author>
                       <http://eg.org/Marlowe>))))
```

book
ex:Tamburlaine
ex:DoctorFaustus

book
ex:Macbeth
ex:Hamlet

Beispielauswertung (2)

```
Filter(?price < 15,
    Join(
      LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
               Bgp(?book <http://eg.org/Titel> ?titel),
               true),
      Union(Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Shakespeare>),
            Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Marlowe>))))
```

book
ex:Hamlet
ex:Macbeth
ex:Tamburlaine
ex:DoctorFaustus

Beispielauswertung (3)

```
Filter(?price < 15,
    Join(
      LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
               Bgp(?book <http://eg.org/Titel> ?titel),
               true),
      Union(Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Shakespeare>),
            Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Marlowe>))))
```

book	price
ex:Hamlet	10.5
ex:Tamburlaine	17
ex:DoctorFaustus	12
ex:RomeoJulia	9

Beispielauswertung (3)

```
Filter( ?price < 15,
    Join(
      LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
               Bgp(?book <http://eg.org/Titel> ?titel),
               true),
      Union(Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Shakespeare>),
            Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Marlowe>))))
```

book	price
ex:Hamlet	10.5
ex:Tamburlaine	17
ex:DoctorFaustus	12
ex:RomeoJulia	9

book	titel
ex:DoctorFaustus	"The Tragical History of Doctor Faustus"

Beispielauswertung (4)

```
Filter(?price < 15,
    Join(
      LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
               Bgp(?book <http://eg.org/Titel> ?titel),
               true),
      Union(Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Shakespeare>),
            Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Marlowe>))))
```

book	price	titel
ex:Hamlet	10.5	
ex:Tamburlaine	17	
ex:DoctorFaustus	12	"The Tragical History of Doctor Faustus"
ex:RomeoJulia	9	

Beispielauswertung (5)

```
Filter( ?price < 15,
    Join(
      LeftJoin(Bgp( ?book <http://eg.org/Price> ?price),
               Bgp( ?book <http://eg.org/Titel> ?titel),
               true),
      Union(Bgp( ?book <http://eg.org/Author>
                       <http://eg.org/Shakespeare>),
            Bgp( ?book <http://eg.org/Author>
                       <http://eg.org/Marlowe>))))
```

book	price	titel
ex:Hamlet	10.5	
ex:Tamburlaine	17	
ex:DoctorFaustus	12	"The Tragical History of Doctor Faustus"

Beispielauswertung (6)

```
Filter(?price < 15,
    Join(
      LeftJoin(Bgp(?book <http://eg.org/Price> ?price),
               Bgp(?book <http://eg.org/Titel> ?titel),
               true),
      Union(Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Shakespeare>),
            Bgp(?book <http://eg.org/Author>
                      <http://eg.org/Marlowe>))))
```

book	price	titel
ex:Hamlet	10.5	
ex:DoctorFaustus	12	"The Tragical History of Doctor Faustus"

Agenda

- Wiederholung
- Auswertung der SPARQL-Algebra
- SPARQL Algebra Transformation
- Operationen für die Modifikatoren
- Abstrakte SPARQL Abfragen
- Zusammenfassung

Formale Algebra Transformation

- Während des Parsens einer Abfrage wird ein Parse Tree erstellt
- Enthält Objekte, die der Abfrage Grammatik entsprechen
- Zur Algebra Transformation wird der Parse Tree traversiert und rekursiv in Algebra Objekte transformiert
- Das Abfragemuster ist ein `GroupGraphPattern` bestehend aus den folgenden Elementen:
  - TriplesBlock
  - Filter
  - OptionalGraphPattern
  - GroupOrUnionGraphPattern
  - GraphGraphPattern

Teile der SPARQL Abfrage Grammatik

```
GroupGraphPattern          ::= '{' TriplesBlock?
                                 ( ( GraphPatternNotTriples
                                   | Filter ) '.' ? TriplesBlock? )*
                               '}'
GraphPatternNotTriples     ::= OptionalGraphPattern
                             | GroupOrUnionGraphPattern
                             | GraphGraphPattern
OptionalGraphPattern       ::= 'OPTIONAL' GroupGraphPattern
GraphGraphPattern          ::= 'GRAPH' VarOrIRIref
                               GroupGraphPattern
GroupOrUnionGraphPattern   ::= GroupGraphPattern ( 'UNION'
                                GroupGraphPattern )*
Filter                     ::= 'FILTER' Constraint
```

Algorithmus zur Algebra Transformation

Algorithm 1 algbr(G)

Eingabe: ein Abfragemuster G
Ausgabe ein SPARQL-Algebra Objekt A

```
1: if G ist TriplesBlock then
2:     A := Bgp(G)
3: else if G ist GroupOrUnionGraphPattern then
4:     A := algbrGroupOrUnionGP(G)
5: else if G ist GraphGraphPattern then
6:     A := algbrGraphGP(G)
7: else if G ist GroupGraphPattern then
8:     A := algbrGroupGP(G)
9: return A
```

Transformation von `GroupOrUnionGraphPattern`

Algorithm 2 algbrGroupOrUnionGP(G)

Eingabe: ein `GroupOrUnionGraphPattern` G
mit Elementen $e_1, \ldots, e_n$

Ausgabe: ein SPARQL-Algebra Ausdruck A

1: **for** $i = 1, \ldots, n$ **do**
2: **if** A ist undefiniert **then**
3: A := algbr(e_i)
4: **else**
5: A := Union(A, algbr(e_i))
6: **return** A

Transformation von `GraphGraphPattern`

Algorithm 3 algbrGraphGP(G)

Eingabe: ein `GraphGraphPattern`

Ausgabe: ein SPARQL-Algebra Ausdruck A

1: **if** G `GRAPH IRI GroupGraphPattern` **then**
2: A := Graph(`IRI`, algbr(GroupGraphPattern))
3: **else if** G `GRAPH Var GroupGraphPattern` **then**
4: A := Graph(`Var`, algbr(GroupGraphPattern))
5: **return** A

Transformation von `GroupGraphPattern`

Algorithm 4 algbrGroupGP(G)

Eingabe: ein `GroupGraphPattern` G = $(e_1, \ldots, e_n)$
Ausgabe: ein SPARQL-Algebra Ausdruck A

```
 1: A := Z { das leere Muster}
 2: F := ∅ { Filter}
 3: for i = 1, ..., n do
 4:     if e_i ist FILTER( f ) then
 5:         F := F ∪ {f}
 6:     else if e_i ist OPTIONAL { P } then
 7:         if algbr(P) ist Filter(F', A') then
 8:             A := LeftJoin(A, A', F')
 9:         else
10:             A := LeftJoin(A, algbr(P), true)
11:     else
12:         A := Join(A, algbr(e_i))
13: if F ≠ ∅ then
14:     A := Filter(⋀_{f∈F} f, A)
15: return A
```

Vereinfachung von Algebra Objekten

- Gruppen mit nur einem Muster (ohne Filter) werden zu $\text{Join}(Z, A)$ und können durch A Ersetzt werden
- Das leere Muster ist die Identität bzgl. Joins:
  - Ersetze $\text{Join}(Z, A)$ durch A
  - Ersetze $\text{Join}(A, Z)$ durch A

Agenda

- Wiederholung
- Auswertung der SPARQL-Algebra
- SPARQL Algebra Transformation
- Operationen für die Modifikatoren
- Abstrakte SPARQL Abfragen
- Zusammenfassung

Operationen zur Darstellung der Modifikatoren

ToList(M)	Erzeugt aus einer Multimenge eine Sequenz mit den gleichen Elementen und gleicher Kardinalität (beliebige Ordnung, Duplikate nicht unbedingt benachbart)
OrderBy(M, Sortierangaben)	Sortiere Lösungen in Ergebnisliste
Distinct(M)	Entferne doppelte Lösungen aus Ergebnisliste
Reduced(M)	Kann doppelte Lösungen aus Ergebnisliste entfernen
Slice(M, o, l)	Beschneide Ergebnisliste auf Abschnitt der Länge l ab Position o
Project(M, Variablenliste)	Beschränke alle Lösungen auf die angegebenen Variablen

Transformation der Modifikatoren

Sei q eine SPARQL Abfrage mit Abfragemuster P und entsprechendem Algebra Object A_P. Wir konstruieren ein Algebra Objekt A_q für q wie folgt:

1. $A_q := \mathsf{ToList}(\mathsf{A_P})$
2. $A_q := \mathsf{OrderBy}(\mathsf{A_q}, (\mathsf{c}_1, \ldots, \mathsf{c}_n))$ wenn q eine ORDER BY Bedingung hat mit den Vergleichsoperatoren $c_1, \ldots, c_n$
3. $A_q := \mathsf{Project}(\mathsf{A_q}, \mathsf{vars})$ wenn die Ausgabeform `SELECT` ist mit `vars` die selektierten Variablen ($*$ alle Variablen)
4. $A_q := \mathsf{Distinct}(\mathsf{A_q})$ wenn die Ausgabeform `SELECT` ist und q `DISTINCT` enthält
5. $A_q := \mathsf{Reduced}(\mathsf{A_q})$ wenn die Ausgabeform `SELECT` ist und q `REDUCED` enthält
6. $A_q := \mathsf{Slice}(\mathsf{A_q}, \mathsf{start}, \mathsf{length})$ wenn die Abfrage `OFFSET start` oder `LIMIT length` enthält mit `start` Default 0 und `length` Default $(|[\![A_q]\!]_{\mathsf{G}}| - \mathtt{start})$

Auswertung der Modifikatoren

Die Algebra Objekte für die Modifikatoren werden rekursiv ausgewertet

- Werte den im Operator enthaltenen Algebra Ausdruck aus
- Wende die dem Modifikator Objekt entsprechende Operation auf die erhaltene Ergebnismenge an

Agenda

- Wiederholung
- Auswertung der SPARQL-Algebra
- SPARQL Algebra Transformation
- Operationen für die Modifikatoren
- Abstrakte SPARQL Abfragen
- Zusammenfassung

Abstrakte SPARQL Abfragen

Definition (Abstrakte SPARQL Abfrage)

Eine abstrakte SPARQL Abfrage ist ein Tripel (E, DS, QF) mit:

- E einem SPARQL Algebra Ausdruck
- DS ein RDF Dataset
- QF eine Ausgabeform

- E erhalten wir durch Algebra Transformation der Abfrage
- DS ist gegeben (Konfiguration des SPARQL Prozessors)
- QF entnehmen wir der Abfrage (`SELECT`, `CONSTRUCT`, `ASK`, `DESCRIBE`)

Auswertung abstrakter SPARQL Abfragen

- Wir können nun eine abstrakte Anfrage auswerten, indem wir E über DS auswerten und dann die Ergebnisse entsprechend QF aufbereiten
  - `SELECT`: Rückgabe aller Lösungen
  - `ASK`: Rückgabe von `true` wenn die Auswertung mindestens eine Lösung ergibt, sonst `false`
  - `CONSTRUCT`: Rückgabe der Instanziierung der Schablone mit allen Lösungen
  - `DESCRIBE`: Nicht spezifizierte Beschreibung der Lösungen

Agenda

- Wiederholung
- Auswertung der SPARQL-Algebra
- SPARQL Algebra Transformation
- Operationen für die Modifikatoren
- Abstrakte SPARQL Abfragen
- Zusammenfassung

Zusammenfassung

- Wir haben gelernt, wie SPARQL Abfragen ausgewertet werden
- Die Abfrage wird in ein Algebra Objekt umgewandelt
- Die Abfragemuster (BGP) generieren Lösungen
- Die weiteren Algebra Operatoren kombinieren Lösungsmengen
- Die Abfrageform bestimmt, was dann mit den berechneten Lösungen gemacht wird
- Auf der Webseite: engl. lecture notes mit weiteren Übungen, Erklärungen, Beispielen, URLs für öffentliche SPARQL Endpunkte, . . .

Ausblick

- Nächste Vorlesung SPARQL 1.1 Features
- Nicht-Query Teile der Spezifikation (Protokoll, Service Descriptions, Update, . . .)
- Dann: Entailment Regimes (SPARQL mit automatischem Schlußfolgern)
- Dann: (Effiziente) Implementierung von SPARQL