



## Semantic Web Grundlagen

### Semantik von SPARQL

Birte Glimm  
Institut für Künstliche Intelligenz | 22. Dez 2011

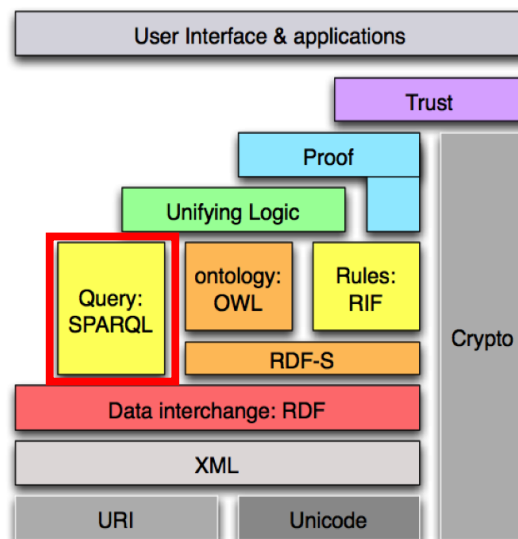
Foliensatz adaptiert von M. Krötzsch. Die nichtkommerzielle Vervielfältigung, Verbreitung und Bearbeitung ist zulässig ( → Lizenz CC-BY-NC).

## 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	<b>SPARQL Semantik</b>	<b>22. Dez</b>
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 Implementierung	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



## 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)
}
```

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

## 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(·) Operator

↔ Bezieht sich auf benachbarte Muster und hat höhere  
Priorität als Konjunktion (linksassoziativ)

## 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)
  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)
}
```

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

## Agenda

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

## 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>

## 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 für <i>leeren Ausdruck</i>
$Graph(\text{varOrIRI}, A)$	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

↪ Wie sollen “Ergebnisse” formal dargestellt werden?

## Berechnung einfacher Graphmuster

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

1. Die Domäne von  $\mu$  ist genau die Menge der Variablen in  $P$
2. Es gibt ein RDF Instanz Mapping  $\sigma$  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  $\mu$

- ▶ wobei die Vielfachheit jedes  $\mu$  der Anzahl der verschiedenen RDF Instanz Mappings  $\sigma$  entspricht

## SPARQL-Ergebnisse

Intuition: Ergebnisse kodieren Tabellen mit Variablenbelegungen

### Ergebnis:

Liste von *Lösungen* (Lösungssequenz)

↪ jede Lösung entspricht einer Tabellenzeile

### Lösung:

Partielle Abbildung (Funktion)

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

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

## Multimengen

### Definition Multimenge

Eine Multimenge über eine Menge  $S$  ist eine totale Funktion

$$M: S \rightarrow \mathbf{N}^+ \cup \{\omega\}$$

- ▶  $\mathbf{N}^+$  die positiven natürlichen Zahlen
- ▶  $\omega > n$  für alle  $n \in \mathbf{N}^+$
- ▶  $M(s)$  ist die Vielfachheit von  $s \in S$
- ▶  $\omega$ : 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:  $\{a, b, b\}$  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" ] .

```

$Bgp(?who \text{ ex:gives } \_ :x . \_ :x \text{ ex:hasTopic } ?what)$

```

μ1: ?who ↦ ex: Birte,      ?what ↦ "SPARQL"
σ1:  _:x ↦ _:a
μ2: ?who ↦ ex: Sebastian, ?what ↦ "DLs and OWL"
σ2:  _:x ↦ _:b

```

Zwei Lösungen jeweils mit Kardinalität 1

## 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  $\llbracket Filter(F, A) \rrbracket_G$ , mit  $F$  einer Filterbedingung und  $A$  einem Algebra Objekt ergibt:

$$\left\{ (\mu, n) \mid M = \llbracket A \rrbracket_G, M(\mu) = n > 0 \text{ and } \llbracket \mu(F) \rrbracket = \text{true} \right\}$$

$\llbracket \mu(F) \rrbracket$  ist das Boolesche Ergebnis der Filterauswertung

## Vereinigung von Lösungen

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

```

μ1: ?x ↦ ex : a, ?y ↦ ex : b      μ1: ?x ↦ ex : a
μ2: ?y ↦ ex : b, ?z ↦ ex : c ✓    μ2: ?y ↦ ex : b ✓
μ1: ?x ↦ ex : a, ?y ↦ ex : b
μ2: ?x ↦ ex : b, ?z ↦ ex : c ✗

```

Vereinigung von zwei kompatiblen Lösungen  $\mu_1$  und  $\mu_2$ :

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

↔ Einfache Intuition: Vereinigung von Tabellenzeilen

## Auswertung von $Join(\cdot)$

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

- Sei  $M_1 = \llbracket A_1 \rrbracket_G$
- Sei  $M_2 = \llbracket A_2 \rrbracket_G$
- Sei  $J(\mu) = \{ (\mu_1, \mu_2) \mid M_1(\mu_1) > 0, M_2(\mu_2) > 0, \mu_1 \text{ und } \mu_2 \text{ sind kompatibel und } \mu = \mu_1 \cup \mu_2 \}$

Die Auswertung  $\llbracket Join(A_1, A_2) \rrbracket_G$  ergibt das Ergebnis

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

## Beispiel zu $Join(\cdot)$

Wir betrachten  $Join(A_1, A_2)$  über einem Graphen  $G$  mit  $\llbracket A_1 \rrbracket_G = M_1$  und  $\llbracket A_2 \rrbracket_G = M_2$  und:

$$M_1 = \{((\mu_1: ?x \mapsto ex : a, ?y \mapsto ex : b), 2),$$

$$((\mu_2: ?x \mapsto ex : a, 1))\}$$

$$M_2 = \{((\mu_3: ?y \mapsto ex : b, ?z \mapsto ex : c, 3))\}$$

$$\mu = ?x \mapsto ex : a, ?y \mapsto ex : b, ?z \mapsto ex : c$$

$$J(\mu) = \{(\mu_1, \mu_3), (\mu_2, \mu_3)\}$$

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

$$= \{(\mu, 9)\}$$

$$n = 2 * 3 + 1 * 3 = 6 + 3 = 9$$

## Auswertung von $Union(\cdot)$

Die Auswertung von  $Union(A_1, A_2)$  über einem Graphen  $G$ , geschrieben  $\llbracket Union(A_1, A_2) \rrbracket_G$ , mit  $A_1, A_2$  Algebra Objekten ergibt:

$$\left\{ (\mu, n) \mid M_1 = \llbracket A_1 \rrbracket_G, M_2 = \llbracket A_2 \rrbracket_G, n = M_1(\mu) + M_2(\mu) > 0 \right\}$$

## 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

$$\blacktriangleright M_1 = \llbracket A_1 \rrbracket_G$$

$$\blacktriangleright M_2 = \llbracket A_2 \rrbracket_G$$

Die Auswertung  $\llbracket LeftJoin(A_1, A_2, F) \rrbracket_G$  von  $LeftJoin(A_1, A_2, F)$  über  $G$  ergibt

$$\llbracket Filter(F, Join(A_1, A_2)) \rrbracket_G \cup$$

$$\left\{ (\mu_1, M_1(\mu_1)) \mid \text{für alle } \mu_2 \text{ mit } M_2(\mu_2) > 0 : \mu_1 \text{ und } \mu_2 \text{ sind} \right.$$

$$\left. \text{inkompatibel oder } \llbracket (\mu_1 \cup \mu_2)(F) \rrbracket = \text{false} \right\}$$

## Auswertung von $Graph(\cdot)$

Die Auswertung von  $Graph(\text{varOrIRI}, A)$  über einem Graphen  $G$ , geschrieben  $\llbracket Graph(\text{varOrIRI}, A) \rrbracket_G$ , mit  $A$  einem Algebra Objekt:

- $\blacktriangleright$  Ist  $\text{varOrIRI}=I$ , für eine IRI  $I$  und enthält das Dataset für  $I$  einen Graphen  $D_I$ , dann ist das Ergebnis  $\llbracket A \rrbracket_{D_I}^{D_I}$
- $\blacktriangleright$  Ist  $\text{varOrIRI}=I$ , für eine IRI  $I$  und enthält das Dataset für  $I$  keinen Graphen  $D_I$ , dann ist das Ergebnis die leere Menge
- $\blacktriangleright$  Ist  $\text{varOrIRI}=v$ , 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 . }
}
```

## 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

```
@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 (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

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 `algr(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` := `algrGroupOrUnionGP(G)`
  - 5: **else if** `G` ist `GraphGraphPattern` **then**
  - 6:   `A` := `algrGraphGP(G)`
  - 7: **else if** `G` ist `GroupGraphPattern` **then**
  - 8:   `A` := `algrGroupGP(G)`
  - 9: **return** `A`
-

## Transformation von GroupOrUnionGraphPattern

---

### Algorithm 2 albrGroupOrUnionGP(G)

---

**Eingabe:** ein GroupOrUnionGraphPattern G  
mit Elementen  $e_1, \dots, e_n$

**Ausgabe:** ein SPARQL-Algebra Ausdruck A

```

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

```

---

## Transformation von GraphGraphPattern

---

### Algorithm 3 albrGraphGP(G)

---

**Eingabe:** ein GraphGraphPattern

**Ausgabe:** ein SPARQL-Algebra Ausdruck A

```

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

```

---

## Transformation von GroupGraphPattern

---

### Algorithm 4 albrGroupGP(G)

---

**Eingabe:** ein GroupGraphPattern  $G = (e_1, \dots, e_n)$

**Ausgabe:** ein SPARQL-Algebra Ausdruck A

```

1: A := Z { das leere Muster}
2: F :=  $\emptyset$  { Filter}
3: for  $i = 1, \dots, n$  do
4:   if  $e_i$  ist FILTER( f ) then
5:     F := F  $\cup$  {f}
6:   else if  $e_i$  ist OPTIONAL { P } then
7:     if albr(P) ist Filter(F', A') then
8:       A := LeftJoin(A, A', F')
9:     else
10:      A := LeftJoin(A, albr(P), true)
11:   else
12:     A := Join(A, albr( $e_i$ ))
13: if F  $\neq \emptyset$  then
14:   A := Filter( $\bigwedge_{f \in F} f$ , A)
15: return A

```

---

## Vereinfachung von Algebra Objekten

- ▶ Gruppen mit nur einem Muster (ohne Filter) werden zu Join(Z, A) und können durch A Ersetzt werden
- ▶ Das leere Muster ist die Identität bzgl. Joins:
  - ▶ Ersetze Join(Z, A) durch A
  - ▶ Ersetze 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)
$OrderBy(M, \text{Sortierangaben})$	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 $o$
$Project(M, \text{Variablenliste})$	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.  $A_q := ToList(A_P)$
2.  $A_q := OrderBy(A_q, (c_1, \dots, c_n))$  wenn  $q$  eine ORDER BY Bedingung hat mit den Vergleichsoperatoren  $c_1, \dots, c_n$
3.  $A_q := Project(A_q, \text{vars})$  wenn die Ausgabeform SELECT ist mit  $\text{vars}$  die selektierten Variablen (\* alle Variablen)
4.  $A_q := Distinct(A_q)$  wenn die Ausgabeform SELECT ist und  $q$  DISTINCT enthält
5.  $A_q := Reduced(A_q)$  wenn die Ausgabeform SELECT ist und  $q$  REDUCED enthält
6.  $A_q := Slice(A_q, \text{start}, \text{length})$  wenn die Abfrage OFFSET  $\text{start}$  oder LIMIT  $\text{length}$  enthält mit  $\text{start}$  Default 0 und  $\text{length}$  Default  $(|[A_q]_G| - \text{start})$

## 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