ulm university universität ulm

Semantic Web Grundlagen

SPARQL Syntax & Intuition

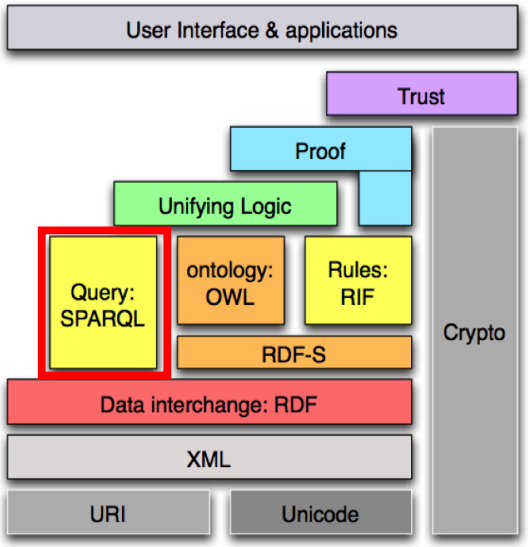
Birte Glimm
Institut für Künstliche Intelligenz | 19. Dez 2011

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



Agenda

- Einleitung und Motivation
- Einfache SPARQL-Anfragen
- Komplexe Graph-Muster in SPARQL
- Filter in SPARQL
- Ausgabeformate in SPARQL
- Modifikatoren in SPARQL
- Zusammenfassung Syntax
- SPARQL Semantik
- Umwandlung von Anfragen in SPARQL-Algebra
- Ausblick

Anfragesprachen für das Semantic Web?

Wie kann auf in RDF oder OWL spezifizierte Informationen zugegriffen werden?

Abfrage von Informationen in RDF(S)

- Einfache Folgerung
- RDF-Folgerung
- RDFS-Folgerung

„Folgt ein bestimmter RDF-Graph aus einem gegebenen?“

Abfrage von Informationen in OWL

- Logisches Schließen

„Folgt eine Subklassen-Beziehung aus einer OWL-Ontologie?“
„Welches sind die Instanzen einer Klasse einer OWL-Ontologie?“

Genügen OWL und RDF nicht?

Selbst OWL ist als Anfragesprache oft zu schwach

- „Welche Zeichenketten in deutscher Sprache sind in der Ontologie angegeben?“
- „Welche Propertys verbinden zwei bestimmte Indiviuen?“
- „Welche Paare von Personen haben eine gemeinsames Elternteil?“

$\rightsquigarrow$ weder in RDF noch in OWL ausdrückbar.

Anforderungen:

- Große Ausdruckstärke zur Beschreibung der gefragten Information
- Möglichkeiten zur Formatierung, Einschränkung und Manipulation der Ergebnisse

Agenda

SPARQL

SPARQL (sprich engl. *sparkle*) steht für
SPARQL Protocol And RDF Query Language

- W3C Spezifikation seit 2008
- Zur Zeit Erweiterung auf SPARQL 1.1
- Anfragsprache zur *Abfrage von Instanzen aus RDF-Dokumenten*
- Große praktische Bedeutung

Teile der SPARQL 1.0 Spezifikation

- Anfragesprache: Thema dieser Vorlesung
- Ergebnisformat: Darstellung von Ergebnissen in XML
- Anfrageprotokoll: Übermittlung von Anfragen und Ergebnissen

Teile der SPARQL 1.1 Spezifikation

- Query: Erweitert die Sprachkonstrukte für SPARQL Abfragen
- Update: zur Modifikation von RDF Graphen (Hinzufügen, Löschen)
- Graph Store HTTP Protocol: HTTP Operationen um eine Menge von Graphen zu verwalten
- Entailment Regimes: Abfragen auch von impliziten Konsequenzen
- Service Description: Methode zum Beschreiben von SPARQL Endpunkten
- Federation Extensions: Ausführung von verteilten Abfragen
- Query Results JSON Format: Abfrageergebnisse in JSON
- Query Results CSV, TSV Format: Komma und Tab separierte Abfrageergebnisse

Einfache Anfragen

Eine einfache Beispielanfrage:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox }
```

- Die Bedingung der WHERE Klausel heisst *query pattern/Abfragemuster*
- Tripel mit Variablen heissen basic graph pattern (BGP)
  - ⇝ BGPs verwenden Turtle Syntax für RDF
  - ⇝ BGPs können Variablen (`?variable` oder `$variable`) enthalten
- Abgekürzte IRIs sind möglich (`PREFIX`)
- Abfrageergebnis für die selektierten Variablen (`SELECT`)

Beispielergebnis

BGP: `{?x foaf:name ?name. ?x foaf:mbox ?mbox}`

```
@prefix foaf: http://xmlns.com/foaf/0.1/ .
_:a foaf:name "Birte Glimm" ;
    foaf:mbox "b.glimm@googlemail.com" ;
    foaf:icqChatID "b.glimm" ;
    foaf:aimChatID "b.glimm" .
_:b foaf:name "Sebastian Rudolph" ;
    foaf:mbox <rudolph@kit.edu> .
_:c foaf:name "Pascal Hitzler" ;
    foaf:aimChatID "phi" .
foaf:icqChatID rdfs:subPropertyOf foaf:nick .
foaf:name rdfs:domain foaf:Person .
```

BGP Matching Ergebnis (Tabelle mit einer Zeile je Ergebnis):

x	name	mbox
_:a	"Birte Glimm"	"b.glimm@googlemail.com"
_:b	"Sebastian Rudolph"	<mailto:rudolph@kit.edu>

Beispielergebnis

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?name ?mbox
WHERE { ?x foaf:name ?name .
        ?x foaf:mbox ?mbox }
```

BGP Matching Ergebnis:

x	name	mbox
_:a	"Birte Glimm"	"b.glimm@googlemail.com"
_:b	"Sebastian Rudolph"	<mailto:rudolph@kit.edu>

Abfrageergebnis:

name	mbox
"Birte Glimm"	"b.glimm@googlemail.com"
"Sebastian Rudolph"	<mailto:rudolph@kit.edu>

Einfache Graph-Muster

Die grundlegenden Anfragemuster sind einfache Graph-Muster oder basic graph patterns (BGPs)

- Menge von RDF-Tripeln in Turtle-Syntax
- Turtle-Abkürzungen (mittels , und ;) zulässig
- Variablen werden durch ? oder $ gekennzeichnet (`?variable` hat gleiche Bedeutung wie `$variable`)
- Variablen zulässig als Subjekt, Prädikat oder Objekt

Zulässig $\neq$ lesbar:

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
SELECT ?rf456df ?ac66sB
WHERE { ?h4dF8Q foaf:name ?rf456df .
        ?h4dF8Q foaf:mbox ?ac66sB }
```

(semantisch äquivalent zur vorherigen Anfrage)

Was bedeuten leere Knoten in SPARQL?

Leere Knoten in Anfragemustern:

- Zulässig als Subjekt oder Objekt
- ID beliebig, aber niemals gleiche ID mehrfach pro Anfrage
- Verhalten sich wie Variablen, die nicht ausgewählt werden können

Leere Knoten in Ergebnissen:

- Platzhalter für unbekannte Elemente
- IDs beliebig, aber eventuell an andere Ergebnisteile gebunden:

subj	Wert
_:a	"zum"
_:b	"Beispiel"

subj	Wert
_:y	"zum"
_:g	"Beispiel"

subj	Wert
_:z	"zum"
_:z	"Beispiel"

Datasets und `FROM (NAMED)`

- Keine `FROM` Klausel notwendug
- Jeder SPARQL Service spezifiziert ein Dataset bestehend aus einem Default Graphen und keinem oder mehr benannten Graphen (named graphs)

Keine `FROM` Klausel
 ⇝ Auswertung bzgl. des Default Graphen
`FROM NAMED` in Kombination mit dem `GRAPH` Schlüsselwort
 ⇝ Auswertung über den benannten Graphen
`FROM` Klausel
 ⇝ Erzeugung eines neuen Default Graphen für die Abfrage

```
SELECT ?name ?mbox
FROM NAMED <http://ex.org/a> <http://ex.org/b>
WHERE { GRAPH ?g
          { ?x foaf:name ?name. ?x foaf:mbox ?mbox }
      }
```


Datentypen

```
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix ex:  <http://example.org/> .
ex:ex1  ex:p  "test" .
ex:ex2  ex:p  "test"^^xsd:string .
ex:ex3  ex:p  "test"@en .
ex:ex4  ex:p  "42"^^xsd:integer .
```

Was liefert eine Anfrage mit folgendem Muster?

```
{ ?subject <http://example.org/p> "test" . }
```

⇝ `ex:ex1` als einziges Ergebnis
⇝ genaue Übereinstimmung der Datentypen gefordert

Aber: Abkürzung für Zahlenwerte möglich

```
{ ?subject <http://example.org/p> 42 . }
```

⇝ Datentyp wird aus syntaktischer Form bestimmt:
`xsd:integer` (42), `xsd:decimal` (42.2), `xsd:double` (1.0e6)

Agenda

- Einleitung und Motivation
- Einfache SPARQL-Anfragen
- Komplexe Graph-Muster in SPARQL
- Filter in SPARQL
- Ausgabeformate in SPARQL
- Modifikatoren in SPARQL
- Zusammenfassung Syntax
- SPARQL Semantik
- Umwandlung von Anfragen in SPARQL-Algebra
- Ausblick

Gruppierende Graph-Muster

Einfache Graph-Muster können durch {. . . } gruppiert werden.

Beispiel:

```
PREFIX ex:  <http://example.org/>
SELECT ?titel ?author
WHERE
{ { ?buch ex:publishedBy <http://springer.com/Verlag>.
    ?buch ex:Titel ?titel . }
  { }
  ?buch ex:Author ?author .
}
```

⇝ Sinnvoll erst bei Verwendung zusätzlicher Konstruktoren

Optionale Muster

Das Schlüsselwort `OPTIONAL` erlaubt die Angabe optionaler Teile eines Musters.

Beispiel:

```
{ ?book ex:publishedBy <http://springer.com/Verlag> .
   OPTIONAL { ?book ex:Titel ?titel . }
   OPTIONAL { ?book ex:Author ?author . }
}
```

⇝ Teile eines Anfrageergebnisses können ungebunden sein:

book	titel	author
<http://ex.org/book1>	"Titel1"	<http://ex.org/author1>
<http://ex.org/book2>	"Titel2"	
<http://ex.org/book3>	"Titel3"	_:a
<http://ex.org/book4>		_:a
<http://ex.org/book5>		

Alternative Muster

Das Schlüsselwort `UNION` erlaubt die Angabe alternativer Teile eines Musters.

Example:

```
{ ?book ex:publishedBy <http://springer.com/Verlag> .
  { ?book ex:Author ?author . } UNION
  { ?book ex:Editor ?author . }
}
```

⇝ Ergebnis entspricht Vereinigung der Ergebnisse mit einer der beiden Bedingungen

Anm.: Gleiche Variablennamen in beiden Teilen von `UNION` beeinflussen sich nicht

Kombination von Optionen und Alternativen (1)

Wie sind Kombinationen von `OPTIONAL` und `UNION` zu verstehen?

```
{ ?book ex:publishedBy  <http://springer.com/Verlag> .
  { ?book ex:Author ?author . } UNION
  { ?book ex:Editor ?author . } OPTIONAL
  { ?author ex:Surname ?name . } }
```

- Vereinigung zweier Muster mit angefügtem optionalem Muster oder
- Vereinigung zweier Muster, wobei das zweite einen optionalen Teil hat?

$\rightsquigarrow$ Erste Interpretation korrekt:

```
{ ?book ex:publishedBy <http://springer.com/Verlag> .
  { { ?book ex:Author ?author . } UNION
    { ?book ex:Editor   ?author . }
  } OPTIONAL { ?author ex:Surname ?name . } }
```


Kombination von Optionen und Alternativen (2)

Allgemeine Regeln

- `OPTIONAL` bezieht sich immer auf genau ein gruppierendes Muster rechts davon.
- `OPTIONAL` und `UNION` sind gleichwertig und beziehen sich auf jeweils alle links davon stehenden Ausdrücke (*linksassoziativ*)

Kombination von Optionen und Alternativen (3)

Beispiel:

```
{ {s1 p1 o1} OPTIONAL {s2 p2 o2} UNION {s3 p3 o3}
             OPTIONAL {s4 p4 o4} OPTIONAL {s5 p5 o5}
}
```

bedeutet:

```
{ { { { {s1 p1 o1} OPTIONAL {s2 p2 o2}
      } UNION {s3 p3 o3}
    } OPTIONAL {s4 p4 o4}
  } OPTIONAL {s5 p5 o5}
}
```


Agenda

Wozu Filter?

Viele Anfragen sind auch mit komplexen Graph-Mustern nicht möglich:

- „Welche Personen sind zwischen 18 und 23 Jahre alt?“
- „Der Nachname welcher Personen enthält einen Bindestrich?“
- „Welche Texte in deutscher Sprache sind in der Ontologie angegeben?“

⇝ Filter als allgemeiner Mechanismus für solche Ausdrucksmittel

Filter in SPARQL

Beispiel:

```
PREFIX ex: <http://example.org/>
SELECT ?book WHERE
  { ?book ex:publishedBy <http://springer.com/Verlag> .
    ?book ex:Price ?price
    FILTER (?price < 35)
  }
```

- Schlüsselwort `FILTER`, gefolgt von Filterausdruck in Klammern
- Filterbedingungen liefern Wahrheitswerte (und ev. auch Fehler)
- Viele Filterfunktionen nicht durch RDF spezifiziert
  ⇝ Funktionen teils aus XQuery/XPath-Standard für XML übernommen

Filterfunktionen: Vergleiche und Arithmetik

Vergleichoperatoren: `<`, `=`, `>`, `<=`, `>=`, `!=`

- Vergleich von Datenliteralen gemäß der jeweils *natürlichen* Reihenfolge
- Unterstützung für numerische Datentypen, `xsd:dateTime`, `xsd:string` (alphabetische Ordnung), `xsd:Boolean` (1 > 0)
- für andere Typen und sonstige RDF-Elemente nur = und `!=` verfügbar
- kein Vergleich von Literalen inkompatibler Typen (z.B. `xsd:string` und `xsd:integer`)

Arithmetische Operatoren: `+`, `-`, `*`, `/`

- Unterstützung für numerische Datentypen
- Verwendung zur Kombination von Werten in Filterbedingungen

Bsp.: `FILTER( ?gewicht/(?groesse * ?groesse) >= 25 )`

Filterfunktionen: Spezialfunktionen für RDF (1)

SPARQL unterstützt auch RDF-spezifische Filterfunktionen:

`BOUND(A)`	`true` falls `A` eine gebundene Variable ist
`isURI(A)`	`true` falls `A` eine URI ist
`isBLANK(A)`	`true` falls `A` ein leerer Knoten ist
`isLITERAL(A)`	`true` falls `A` ein RDF-Literal ist
`STR(A)`	lexikalische Darstellung (`xsd:string`) von RDF-Literalen oder URIs
`LANG(A)`	Sprachcode eines RDF-Literals (`xsd:string`) oder leerer String falls kein Sprachcode
`DATATYPE(A)`	Datentyp-URI eines RDF-Literals (`xsd:string` bei ungetypten Literalen ohne Sprachangabe)

Filterfunktionen: Spezialfunktionen für RDF (2)

Weitere RDF-spezifische Filterfunktionen:

`sameTERM(A,B)`	`true`, falls `A` und `B` dieselben RDF-Terme sind
`langMATCHES(A,B)`	`true`, falls die Sprachangabe `A` auf das Muster `B` passt
`REGEX(A,B)`	`true`, falls in der Zeichenkette `A` der reguläre Ausdruck `B` gefunden werden kann

Beispiel:

```
PREFIX ex: <http://example.org/>
SELECT ?book WHERE
  { ?book ex:Review ?text .
    FILTER ( langMATCHES( LANG(?text), "de") )
  }
```


Filterfunktionen: Boolesche Operatoren

Filterbedingungen können mit Booleschen Operatoren verknüpft werden: `&&`, `||`, `!`

Teilweise auch durch Graph-Muster ausdrückbar:

- Konjunktion entspricht Angaben mehrerer Filter
- Disjunktion entspricht Anwendung von Filtern in alternativen Mustern

Agenda

Ausgabeformatierung mit SELECT

Bisher waren alle Ergebnisse Tabellen: Ausgabeformat `SELECT`

Syntax: `SELECT <Variablenliste>` oder `SELECT *`

Vorteil

Einfache sequentielle Abarbeitung von Ergebnissen

Nachteil

Struktur/Beziehungen der Objekte im Ergebnis nicht offensichtlich

Ausgabeformatierung mit CONSTRUCT

Kodierung von Ergebnissen in RDF-Graphen: Ausgabeformat `CONSTRUCT`

Syntax: `CONSTRUCT <RDF-Schablone in Turtle>`

```
PREFIX ex: <http://example.org/>
CONSTRUCT { ?person  ex:mailbox  ?email .
            ?person  ex:telefon  ?telefon . }
WHERE { ?person  ex:email   ?email .
        ?person  ex:tel     ?telefon . }
```

Vorteil

Stukturiertes Ergebnis mit Beziehungen zwischen Elementen

Nachteile

- Sequentielle Abarbeitung von Ergebnissen erschwert
- Keine Behandlung von ungebundenen Variablen

`CONSTRUCT` Schablonen mit Leeren Knoten

```
@prefix  foaf:  <http://xmlns.com/foaf/0.1/> .
_:a  foaf:firstname  "Alice" ;  foaf:surname "Hacker" .
_:b  foaf:firstname  "Bob" ;  foaf:surname    "Hacker" .

PREFIX foaf:    <http://xmlns.com/foaf/0.1/>
PREFIX vcard:   <http://www.w3.org/2001/vcard-rdf/3.0#>
CONSTRUCT { ?x  vcard:N _:v .
     _:v vcard:givenName ?gname ; vcard:familyName ?fname }
WHERE { ?x foaf:firstname ?gname . ?x foaf:surname  ?fname  }
```

Erzeugt den folgenden RDF Graphen:

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
_:v1 vcard:N  _:x .
_:x vcard:givenName  "Alice" ; vcard:familyName "Hacker" .
_:v2 vcard:N  _:z .
_:z vcard:givenName  "Bob" ; vcard:familyName "Hacker" .
```


Weitere Formate: ASK und DESCRIBE

SPARQL unterstützt zwei weitere Ausgabeformate:

- `ASK` prüft nur, ob es Ergebnisse gibt, keine Parameter
- `DESCRIBE` (informativ) liefert zu jeder gefundenen URI eine RDF-Beschreibung (anwendungsabhängig)

```
DESCRIBE ?x WHERE { ?x <http://ex.org/employeeId> "1234" }
```

Kann als Ergebnis z.B. die folgende Ausgabe haben (ohne Prefix Deklarationen):

```
_:a      exOrg:employeeId    "1234" ;
         foaf:mbox_sha1sum   "ABCD1234" ;
         vcard:N
         [ vcard:Family       "Smith" ;
           vcard:Given        "John"  ] .
foaf:mbox_sha1sum  rdf:type  owl:InverseFunctionalProperty .
```


Agenda

Wozu Modifikatoren?

Bisher nur grundsätzliche Formatierungseinstellungen für Ergebnisse:

- Wie kann man definierte Teile der Ergebnismenge abfragen?
- Wie werden Ergebnisse geordnet?
- Können wiederholte Ergebniszeilen sofort entfernt werden?

$\rightsquigarrow$ Modifikatoren der Lösungssequenz (*solution sequence modifiers*)

Ergebnisse Sortieren

Sortierung von Ergebnissen mit Schlüsselwort `ORDER BY`

```
SELECT ?book ?price
WHERE { ?book <http://example.org/Price> ?price . }
ORDER BY ?price
```

- Sortierung wie bei Filter-Vergleichoperatoren,
- Sortierung von URIs alphabetisch als Zeichenketten
- Reihenfolge zwischen unterschiedlichen Arten von Elementen:
  Ungebundene Variable < leere Knoten < URIs < RDF-Literale
- Nicht jede Möglichkeit durch Spezifikation definiert

Ergebnisse sortieren

Weitere mögliche Angaben:

- `ORDER BY DESC(?price)`: absteigend
- `ORDER BY ASC(?price)`: aufsteigend, Voreinstellung
- `ORDER BY DESC(?price), ?titel`: hierarchische Ordnungskriterien

LIMIT, OFFSET und DISTINCT

Einschränkung der Ergebnismenge:

- `LIMIT`: maximale Anzahl von Ergebnissen (Tabellenzeilen)
- `OFFSET`: Position des ersten gelieferten Ergebnisses
- `SELECT DISTINCT`: Entfernung von doppelten Tabellenzeilen

```
SELECT DISTINCT ?book ?price
WHERE { ?book <http://example.org/Price> ?price . }
ORDER BY ?price  LIMIT 5  OFFSET 25
```

$\rightsquigarrow$ `LIMIT` und `OFFSET` nur mit `ORDER BY` sinnvoll!

Kombination von Modifikatoren

Reihenfolge bei Abarbeitung von Modifikatoren:

1. Sortierung gemäß `ORDER BY`
2. Entfernung der nicht ausgewählten Variablen
3. Entfernung doppelter Ergebnisse (`DISTINCT`)
4. Enfernung der ersten `OFFSET` Ergebnisse
5. Entfernung aller Ergebnisse bis auf `LIMIT`

⇝ Sortierung auch nach nicht ausgewählten Variablen möglich

⇝ `ORDER BY` nicht nur für `SELECT` relevant

Agenda

Vorgestellte SPARQL-Merkmale im Überblick

Grundstruktur
`PREFIX`
`WHERE`

Ausgabeformate
`SELECT`
`CONSTRUCT`
`ASK`
`DESCRIBE`

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

Filter
`BOUND`
`isURI`
`isBLANK`
`isLITERAL`
`STR`
`LANG`
`DATATYPE`
`sameTERM`
`langMATCHES`
`REGEX`

Modifikatoren
`ORDER BY`
`LIMIT`
`OFFSET`
`DISTINCT`

SPARQL 1.1 Erweiterungen

SPARQL 1.1 vorerst ausgelassen

- Aggregate
- Subqueries
- Negation
- Ausdrücke in der SELECT Klausel
- Property Paths
- Assignment/Zuweisung
- CONSTRUCT Kurzform
- Weitere Funktionen und Operatoren

⇝ Vorlesung 16: SPARQL 1.1

Nun erst einmal Semantik!

Agenda

- Einleitung und Motivation
- Einfache SPARQL-Anfragen
- Komplexe Graph-Muster in SPARQL
- Filter in SPARQL
- Ausgabeformate in SPARQL
- Modifikatoren in SPARQL
- Zusammenfassung Syntax
- SPARQL Semantik
- Umwandlung von Anfragen in SPARQL-Algebra
- Ausblick

Wozu Semantik?

Bisher lediglich informelle Darstellung von SPARQL

- Anwender: „Welche Antworten kann ich auf meine Anfrage erwarten?“
- Entwickler: „Wie genau soll sich meine SPARQL-Implementierung verhalten?“
- Hersteller: „Ist mein Produkt bereits SPARQL-konform?“

$\rightsquigarrow$ Formale Semantik schafft (hoffentlich) Klarheit . . .

Semantik von Anfragesprachen (1)

Semantik formaler Logik (siehe Vorlesung 5):

- Modelltheoretische Semantik: Welche Interpretationen erfüllen eine Wissensbasis?
- Beweistheoretische Semantik: Welche Ableitungen aus einer Wissenbasis sind zulässig?
- . . .

Semantik von Programmiersprachen:

- Axiomatische Semantik: Welche logischen Aussagen gelten für ein Programm?
- Operationale Semantik: Wie wirkt sich die Abbarbeitung eines Programms aus?
- Denotationelle Semantik: Wie kann ein Programm als Eingabe/Ausgabe-Funktion abstrakt dargestellt werden?

Was tun mit Anfragesprachen?

Semantik von Anfragesprachen (2)

Anfragefolgerung (*query entailment*)

- Anfrage als Beschreibung zulässiger Anfrageergebnisse
- Datenbasis als Menge logischer Annahmen (Theorie)
- Ergebnis als logische Schlussfolgerung

Bsp.: OWL DL und RDF(S) als Anfragesprachen, konjunktive Anfragen

Anfragealgebra

- Anfrage als Rechenvorschrift zur Ermittlung von Ergebnissen
- Datenbasis als Eingabe
- Ergebnis als Ausgabe

Bsp.: Relationale Algebra für SQL, SPARQL-Algebra

Agenda

- Einleitung und Motivation
- Einfache SPARQL-Anfragen
- Komplexe Graph-Muster in SPARQL
- Filter in SPARQL
- Ausgabeformate in SPARQL
- Modifikatoren in SPARQL
- Zusammenfassung Syntax
- SPARQL Semantik
- Umwandlung von Anfragen in SPARQL-Algebra
- Ausblick

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

```
{ 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)
}
```

4. Kombination alternativer Muster mit Union(·) Operator

⇝ 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)
}
```

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

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

- Einleitung und Motivation
- Einfache SPARQL-Anfragen
- Komplexe Graph-Muster in SPARQL
- Filter in SPARQL
- Ausgabeformate in SPARQL
- Modifikatoren in SPARQL
- Zusammenfassung Syntax
- SPARQL Semantik
- Umwandlung von Anfragen in SPARQL-Algebra
- Ausblick

Zusammenfassung

- Wir haben die SPARQL 1.0 Features durch Beispiele kennengelernt
  - Grundlegende Strukturen (Prefixe, Muster)
  - Ausgabeformate
  - Einfache und komplexe Muster (Alternativen, Optionale Teile, Gruppen)
  - Filter
  - Modifikatoren
- Semantik definiert durch Übersetzung in SPARQL Algebra
- Bisher nur informell eingeführt

Ausblick

Offene Fragen

- Wie können wir SPARQL Algebra Objects auswerten?
- Wie funktioniert die systematische Algebra Übersetzung?
- Wie sind die neuen SPARQL 1.1 Features definiert?
- Wie funktioniert das SPARQL Protokoll?
- Wie können wir auch implizite Konsequenzen unter RDF Schema oder OWL Semantik abfragen?
- Wie schwer ist SPARQL (mit Entailment) zu implementieren?
- Welche Implementierungstechniken können wir anwenden?
- ...