ulm university universität ulm

Semantic Web Grundlagen

SPARQL 1.1

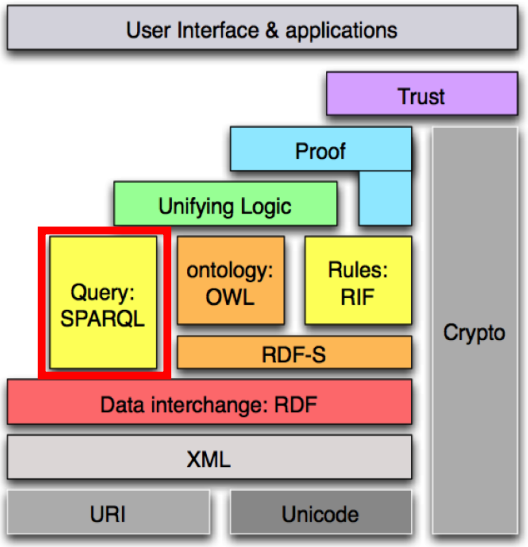
Birte Glimm
Institut für Künstliche Intelligenz | 22. Dez 2011

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



Agenda

- Wiederholung
- SPARQL 1.1 Abfrage Erweiterung
  - Ausdrücke in der Selektion & Bindungen
  - Aggregate
  - Subqueries
  - Property Paths
  - Negation
- SPARQL Protokoll
- SPARQL Update
- Service Descriptions

Beispielmuster

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


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
```


Entsprechung des Musters zur Grammatik

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

```
{ TriplesBlock
  Filter
  OPTIONAL { TriplesBlock }
  { TriplesBlock } UNION
  { TriplesBlock } }
```

```
{ TriplesBlock
  Filter
  OptionalGraphPattern
  GroupOrUnionGraphPattern }
```

`GroupGraphPattern`

Transformation von `GroupGraphPattern`

Algorithm 1 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
```

Übersetzung in SPARQL-Algebra

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


Vereinfachung der SPARQL-Algebra

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


Bedeutung der SPARQL Algebra Operationen

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

Agenda

- Wiederholung
- SPARQL 1.1 Abfrage Erweiterung
  - Ausdrücke in der Selektion & Bindungen
  - Aggregate
  - Subqueries
  - Property Paths
  - Negation
- SPARQL Protokoll
- SPARQL Update
- Service Descriptions

Ausdrücke in der Selektion & Bindungen

Lösungen können durch berechnete Ausdrücke ergänzt werden, wobei eine Zuweisung durch `(Ausdruck AS ?var)` erfolgt:

- `BIND` innerhalb einer Gruppe eines Abfragemusters
- Ausdrücke in der `SELECT` Klausel
- Ausdrücke in der `GROUP BY` Klausel
- `BINDINGS` fügt Lösungen zu den berechneten Lösungen hinzu

Beispiel BIND (ohne Präfix Deklaration)

Daten

```
ex:Buch ex:Titel "SPARQL Tutorial" ; ex:Preis 42 ;
ex:Rabatt 10 .
```

Abfrage

```
SELECT ?titel ?preis WHERE
{ ?b ex:Titel ?titel; ex:Preis ?p ; ex:Rabatt ?r
 BIND ((?p-?r) AS ?preis) }
```

Ergebnis

$?titel \mapsto$ "SPARQL Tutorial", $?preis \mapsto 32$

$\rightsquigarrow$ Algebra: Extend(Bgp(...), ?preis, (?p-?r))

Beispiel Ausdrücke (ohne Präfix Deklaration)

Daten

```
ex:Buch ex:Titel "SPARQL Tutorial" ; ex:Preis 42 ;
ex:Rabatt 10 .
```

Abfrage

```
SELECT ?titel ((?p-?r) AS ?preis) WHERE
{ ?b ex:Titel ?titel; ex:Preis ?p ; ex:Rabatt ?r }
```

Ergebnis

$?titel \mapsto$ "SPARQL Tutorial", $?preis \mapsto 32$

$\rightsquigarrow$ Algebra: Extend(Bgp(...), ?preis, (?p-?r))

Beispiel BINDINGS

Daten

```
ex:Buch1 ex:Titel "SPARQL Tutorial".
ex:Buch2 ex:Titel "SemWeb".
```

Abfrage

```
SELECT ?titel WHERE
{ ?b ex:Titel ?titel }
BINDINGS ?b { (ex:Buch1) }
```

Ergebnis

$?titel \mapsto$ "SPARQL Tutorial"

$\rightsquigarrow$ Bindungen werden konjunktiv (join) verknüpft

Agenda

- Wiederholung
- SPARQL 1.1 Abfrage Erweiterung
  - Ausdrücke in der Selektion & Bindungen
  - Aggregate
  - Subqueries
  - Property Paths
  - Negation
- SPARQL Protokoll
- SPARQL Update
- Service Descriptions

Aggregate

- Aggregate erlauben die Gruppierung von Lösungen und Berechnung von Werten über die Gruppen

Beispiel

```
SELECT    (COUNT(?student) AS ?c), ?vorlesung
WHERE     { ?student ex:besucht ?vorlesung }
GROUP BY  ?vorlesung
HAVING    ?c > 5
```

- `GROUP BY` gruppiert die Lösungen (hier in Studenten die die gleiche Vorlesung besuchen)
- `COUNT` ist ein Aggregat, welches die Anzahl der Lösungen in einer Gruppe zählt (hier die Anzahl der Studenten in der jeweiligen Vorlesung)
- `HAVING` beschränkt aggregierte Werte

Aggregate in SPARQL 1.1

SPARQL 1.1 unterstützt die folgenden Aggregate, die jeweils über den Werten einer Gruppe ausgewertet werden:

- `COUNT` – Zählen der Lösungen
- `MIN` – Finden des Minimalwerts
- `MAX` – Finden des Maximalwerts
- `SUM` – Summieren der Werte
- `AVG` – Bilden des Durchschnitts
- `GROUP_CONCAT` – Stringkonkatenation, Bsp.: `GROUP_CONCAT(?x ; separator=", ")`
- `SAMPLE` – Wählen eines beliebigen Wertes

Beispiel Aggregate

```
ex:Paul ex:hatNote 2.0 .
ex:Paul ex:hatNote 3.0 .
ex:Mary ex:hatNote 2.0 .
ex:Peter ex:hatNote 3.5 .

SELECT ?student (AVG(?note) as ?avg)
WHERE { ?student ex:hatNote ?note }
GROUP BY ?student HAVING (?avg > 2.0)
```

Agenda

- Wiederholung
- SPARQL 1.1 Abfrage Erweiterung
  - Ausdrücke in der Selektion & Bindungen
  - Aggregate
  - Subqueries
  - Property Paths
  - Negation
- SPARQL Protokoll
- SPARQL Update
- Service Descriptions

Subqueries

Abfrage

```
SELECT ?name WHERE {
  ?x foaf:name ?name .
  { SELECT ?x (COUNT(*) AS ?count)
    WHERE { ?x foaf:knows ?y . }
    GROUP BY ?x
    HAVING (?count = 3)
  }
}
```

- Ergebnisse der inneren Abfrage werden kunjunktiv (join) mit den Ergebnissen der äußeren Abfrage verknüpft

Agenda

- Wiederholung
- SPARQL 1.1 Abfrage Erweiterung
  - Ausdrücke in der Selektion & Bindungen
  - Aggregate
  - Subqueries
  - Property Paths
  - Negation
- SPARQL Protokoll
- SPARQL Update
- Service Descriptions

Reguläre Ausdrücke in Mustern

Property Paths werden mittels regulärer Ausdrücke über Prädikaten gebildet

- Pfade bestimmter Länge: `?s ex:p {2, 4} ?o`
- Pfade mit unbestimmter Länge: `?s ex:p +` ?o, `?s ex:p *` `?o`
- Alternative Pfade: `?s` (`ex:p` $_1$|`ex:p` $_2$) `?o`
- Negation von Pfaden: `?s !ex:p ?o`
- Inverse Pfade: `?s ^ex:p ?o` entspricht `?o ex:p ?s`
- Verknüpfung von Pfaden: `?s ex:p` $_1$ / `ex:p` $_2$ `?o`

Property Pfade Beispiel

Abfrage 1

```
PREFIX ...
SELECT ?xName WHERE {
  ?x rdf:type foaf:Person .
  ?x foaf:name ?xName
  ?x foaf:knows{2}/foaf:name "Bill Gates" .
}
```

Abfrage 2

```
PREFIX ...
SELECT ?s WHERE {
  ?s  rdf:type  ?type .
  ?type rdfs:subClassOf* ex:SomeClass .
}
```


Property Pfade Definition

Definition (Property Pfade)

- Jede IRI ist ein *Property Pfad*.
- Seien $\mathtt{I}_1, \ldots, \mathtt{I}_\ell$ IRIs mit $1 \leq k \leq \ell$, dann ist !($\mathtt{I}_1 \mid \ldots \mid \mathtt{I}_k \mid \hat{}\mathtt{I}_{k+1} \mid \ldots \mid \hat{}\mathtt{I}_\ell$) ein *Property Pfad*.
- Seien $\mathtt{P}$, $\mathtt{P}_1$ und $\mathtt{P}_2$ Property Pfade und seien $n, m \in \mathbf{N}_0$ natürliche Zahlen mit $m > n$, dann sind
  - $\hat{}\mathtt{P}$,
  - $\mathtt{P?}$,
  - $\mathtt{P*}$,
  - $\mathtt{P+}$,
  - $\mathtt{P\{n\}}$,
  - $\mathtt{P\{n,\ m\}}$,
  - $\mathtt{P\{n,\}}$,
  - $\mathtt{P\{,\ m\}}$,
  - $\mathtt{P}_1 \mid \mathtt{P}_2$ und
  - $\mathtt{P}_1 \,/\, \mathtt{P}_2$

  Property Pfade.

Property Pfade Algebra

- Property Pfade werden, soweit möglich, in bestehende SPARQL Konstrukte übersetzt
- Es sind aber auch vier neue Operatoren nötig
  1. NegatedPropertySet(X, {$\text{iri}_1 \ldots \text{iri}_n$},Y)
  2. ZeroOrMorePath(X, path, Y)
  3. OneOrMorePath(X, path, Y)
  4. ZeroLengthPath(X, path, Y)

Agenda

- Wiederholung
- SPARQL 1.1 Abfrage Erweiterung
  - Ausdrücke in der Selektion & Bindungen
  - Aggregate
  - Subqueries
  - Property Paths
  - Negation
- SPARQL Protokoll
- SPARQL Update
- Service Descriptions

Negation in Abfragen

- Zwei Formen der Negation mit konzeptuellen und kleinen semantischen unterschieden
  1. Testen auf Nicht-Erfüllung eines Musters
  2. Entfernung von Lösungen die ein Muster erfüllen

1. Filter

```
SELECT ?x WHERE {
  ?x rdf:type foaf:Person .
  FILTER NOT EXISTS { ?x foaf:name ?name }
}
```

2. Minus

```
SELECT ?x WHERE {
  ?x rdf:type foaf:Person .
  MINUS { ?x foaf:name ?name }
}
```


Auswertung Negation via Filter

Daten

```
_:x rdf:type foaf:Person .
_:x foaf:name "Peter" .
_:y rdf:type foaf:Person .
```

Abfrage Muster

```
{ ?x rdf:type foaf:Person .
  FILTER NOT EXISTS { ?x foaf:name ?name } }
```

1. $[\![$ Bgp(1. Muster) $]\!]_G$: μ_1: ?x $\mapsto$ _:x, μ_2: ?x $\mapsto$ _:y
2. Für jede Lösung, instanziiere das 2. Muster
   - Lösung wird entfernt, wenn die Instanziierung matched (μ_1)
   - sonst bleibt die Lösung (μ_2)

Auswertung Negation via Minus

Daten

```
_:x rdf:type foaf:Person .
_:x foaf:name "Peter" .
_:y rdf:type foaf:Person .
```

Abfrage Muster

```
{ ?x rdf:type foaf:Person .
  MINUS { ?x foaf:name ?name } }
```

$[\![$ Bgp(1. Muster) $]\!]_G$: $\Omega_1 = \{\mu_1$: ?x $\mapsto$ _:x, μ_2: ?x $\mapsto$ _:y$\}$
$[\![$ Bgp(2. Muster) $]\!]_G$: $\Omega_2 = \{\mu_3$: ?x $\mapsto$ _:x, ?name $\mapsto$ "Peter"$\}$
$[\![$ Minus(Ω_1, Ω_2) $]\!]_G$: $\Omega = \{\mu \mid \mu \in \Omega_1$ und $\forall \mu' \in \Omega_2$
μ und μ' sind inkompatibel oder
dom$(\mu) \cap$ dom$(\mu') = \emptyset\}$

$\mu_1 \notin \Omega$: μ_1 kompatibel mit μ_3 und nicht-disjunkte Domänen
$\mu_2 \in \Omega$: μ_2 inkompatibel mit μ_3

Unterschiede Minus und Filter Negation

Daten

```
ex:a ex:b ex:c .
```

Abfrage Muster

```
{ ?s ?p ?o FILTER NOT EXISTS { ?x ?y ?z } }
```

- Filter Muster matched immer (Variablen disjunkt) $\rightsquigarrow$ jede Lösung wird entfernt

Abfrage Muster

```
{ ?s ?p ?o MINUS { ?x ?y ?z } }
```

- Minus entfernt keine Lösungen, da Domänen aller Lösungen disjunkt

Agenda

- Wiederholung
- SPARQL 1.1 Abfrage Erweiterung
  - Aggregate
  - Subqueries
  - Property Paths
  - Ausdrücke in der Selektion
  - Bindungen
- SPARQL Protokoll
- SPARQL Update
- Service Descriptions

SPARQL Protokoll

- Regelt wie Abfragen an einen SPARQL Endpunkt im Web geschickt werden können und wie Resultate zurückgegeben werden
- Regelt wie Fehler kommuniziert werden
- Query
  - `GET` Anfrage etc. Teil der URI: `http://server/endpoint1?query=...`
  - `POST` Anfrage im Body der HTTP Anfrage z.B. via HTML Form
- Update
  - `http://server/endpoint2?update=...`
  - `POST` mit Content-Type application/sparql-update
  - `POST` via HTML Form
- Query und Update sind separate Services

Graph Store HTTP Protokoll

- RESTful Graph Management
- REST Eigenschaften
  - Adressierbarkeit
  - Unterschiedliche Repräsentationen (verschiedene Darstellungsformen für eine Ressource z.B. HTML, JSON)
  - Zustandslosigkeit (Anfrage enthält alle benötigeten Informationen)
  - Anbieten von bestimmten Operationen für alle Ressourcen (z.B. GET, POST, PUT und DELETE)
  - Verwendung von Hypermedia (z.B. HTML, XML)
- Nutzung von `GET`, `PUT`, `POST`, `DELETE` zur Kommunikation von Änderungen

Agenda

- Wiederholung
- SPARQL 1.1 Abfrage Erweiterung
  - Aggregate
  - Subqueries
  - Property Paths
  - Ausdrücke in der Selektion
  - Bindungen
- SPARQL Protokoll
- SPARQL Update
- Service Descriptions

SPARQL Update

- Zur Manipulation von Graphen
- Basieren auf der Idee eines Graph-Stores (Quads)
  - Hinzufügen und Löschen von Graphen
  - Hinzufügen und Löschen von Tripeln in Graphen
- `LOAD`, `DROP`, `CREATE`
- `INSERT`, `DELETE` für Daten/Tripel
- Keine Transaktionen, eine Anfrage kann mehrere atomare Teilanfragen beinhalten

Beispiel Abfrage

```
DELETE { ?person foaf:givenName "Bill" }
INSERT { ?person foaf:givenName "William" }
WHERE {
  ?person a foaf:Person .
  ?person foaf:givenName "Bill"
}
```


Agenda

- Wiederholung
- SPARQL 1.1 Abfrage Erweiterung
  - Aggregate
  - Subqueries
  - Property Paths
  - Ausdrücke in der Selektion
  - Bindungen
- SPARQL Protokoll
- SPARQL Update
- Service Descriptions

Service Descriptions

- Methode und Vokabular zur Beschreibung von SPARQL Endpunkten
- Client/Benutzer kann Informationen über den SPARQL Service abfragen z.B.
  - unterstütze Erweiterungsfunktionen,
  - das verwendete Dataset oder
  - welche Methoden zum automatischen Schlussfolgern verwendet werden

Service Descriptions Beispiel

HTTP Anfrage

```
GET /sparql/ HTTP/1.1
Host: www.example.org
Accept: text/turtle
```

Mögliche Antwort (Beginn)

```
HTTP/1.1 200 OK
Date: Fri, 09 Oct 2009 17:31:12 GMT
Server: Apache/1.3.29 (Unix) PHP/4.3.4 DAV/1.0.3
Connection: close
Content-Type: text/turtle

@prefix sd: <http://www.w3.org/ns/sparql-service-description#> .
@prefix ent: <http://www.w3.org/ns/entailment/> .
@prefix prof: <http://www.w3.org/ns/owl-profile/> .
@prefix scovo: <http://purl.org/NET/scovo#> .
@prefix void: <http://rdfs.org/ns/void#> .

...
```

Service Descriptions Beispiel

Mögliche Antwort (Fortsetzung)

```
...
[] a sd:Service ;
    sd:endpoint <http://www.example.org/sparql/> ;
    sd:supportedLanguage sd:SPARQL11Query ;
    sd:resultFormat <http://www.w3.org/ns/formats/RDF_XML>,
                    <http://www.w3.org/ns/formats/Turtle> ;
    sd:extensionFunction <http://example.org/Distance> ;
    sd:feature sd:DereferencesURIs ;
    sd:defaultEntailmentRegime ent:RDFS ;
    sd:defaultDatasetDescription [
        a sd:Dataset ;
        sd:defaultGraph [
            a sd:Graph ;
            void:triples 100
        ] ;
        sd:namedGraph [
            a sd:NamedGraph ;
            sd:name <http://www.example.org/named-graph> ;
            sd:entailmentRegime ent:OWL-RDF-Based ;
            sd:supportedEntailmentProfile prof:RL ;
            sd:graph [
                a sd:Graph ;
                void:triples 2000
            ]
        ]
    ] .
<http://example.org/Distance> a sd:Function .
```


Zusammenfassung

- Wir haben die neuen Features von SPARQL 1.1 Query kennengelernt
- SPARQL 1.1 noch kein Standard (Last Call Phase)
- SPARQL UPDATE ermöglicht das Modifizieren von Graphen über Anfragen
- Das Protokoll regelt wie Client und Server kommunizieren können
- Service Descriptions beschreiben einen SPARQL Service (maschinenlesbar)
- Nicht betrachtet: Weitere Ergebnisformate: JSON, CVS, TSV

Ausblick:

- Entailment Regimes: SPARQL mit automatischen Schlussfolgern

Öffentliche SPARQL Endpunkte

DBPedia strukturierte Wikipedia Daten (> 100 Million Tripel): `http://dbpedia.org/sparql`

DBTune 14 Milliraden RDF Tripel über Musik `http://dbtune.org/jamendo/store/user/query`

CKAN Dataset Repository mit SPARQL Service `http://semantic.ckan.net/` `http://semantic.ckan.net/snorql/`

Linked Movie Database `http://data.linkedmdb.org/` und `http://data.linkedmdb.org/sparql`

SPARQL Editor mit Beispielen zu Weltraum Daten `http://api.talis.com/stores/space/items/tutorial/spared.html`

Semantic Web Dog Food Informationen über Autoren, Veröffentlichungen und Konferenzen `http://data.semanticweb.org/snorql`