TECHNISCHE
UNIVERSITÄT
DRESDEN

FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

SPARQL Algebra

Sebastian Rudolph

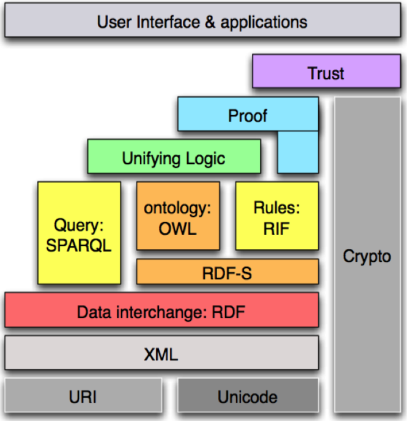
Dresden, May 9

DRESDEN
concept
Exzellenz aus
Wissenschaft
und Kultur

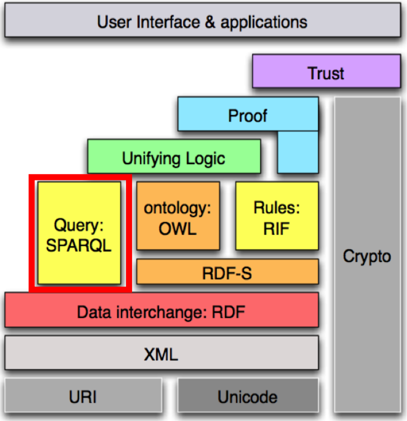
Content

Overview & XML	11 APR DS5
Introduction into RDF	11 APR DS6
RDFS – Syntax & Intuition	16 APR DS6
Tutorial 1	23 APR DS6
RDFS – Semantics	25 APR DS5
RDFS Rule-based Reasoning	25 APR DS6
Tutorial 2	30 APR DS6
SPARQL – Syntax & Intuition	02 MAY DS5
SPARQL – Semantics	02 MAY DS6
SPARQL Algebra	09 MAY DS5
Tutorial 3	09 MAY DS6
OWL – Syntax & Intuition	14 MAY DS6
OWL & Description Logics	16 MAY DS5
OWL 2	16 MAY DS6
Tutorial 4	23 MAY DS5
Tableau I	23 MAY DS6
Tableau II	30 MAY DS5
Tutorial 5	30 MAY DS6
Hypertableau I	4 JUN DS6
Hypertableau II	6 JUN DS5
Tutorial 6	6 JUN DS6
SPARQL 1.1	18 JUN DS6
SPARQL Entailment	20 JUN DS5
Tutorial 7	20 JUN DS6
OWL & Rules	25 JUN DS6
Ontology Editing	27 JUL DS5
Ontology Engineering	27 JUL DS6
Tutorial 8	2 JUL DS6
Linked Data & Applications	4 JUL DS5
Q&A Session	9 JUL DS6
Q&A Session	11 JUL DS5

The SPARQL Query Language



The SPARQL Query Language



Agenda

1. Recap
2. Evaluation of the SPARQL Algebra
3. SPARQL Algebra Transformation
4. Operators for the Modifiers
5. Summary

Agenda

1. Recap
2. Evaluation of the SPARQL Algebra
3. SPARQL Algebra Transformation
4. Operators for the Modifiers
5. Summary

Recap: Introduced SPARQL Features

Basic Structure
PREFIX
WHERE

Graph Patterns
Basic Graph Patterns
{...}
OPTIONAL
UNION

Filter
BOUND
isURI
isBLANK
isLITERAL
STR
LANG
DATATYPE
sameTERM
langMATCHES
REGEX

Modifiers
ORDER BY
LIMIT
OFFSET
DISTINCT

Output Formats
SELECT
CONSTRUCT
ASK
DESCRIBE

Translation into SPARQL Algebra

```
{ ?book ex:price ?price .
  FILTER (?price < 15)
  OPTIONAL { ?book ex:title ?title }
  { ?book ex:author ex:Shakespeare } UNION
  { ?book ex:author ex:Marlowe }
}
```

Semantics of a SPARQL query:

1. Transformation of the query into an algebra expression
2. Evaluation of the algebra expression

Translation into SPARQL Algebra

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

Attention: Filters apply to the whole group in which they occur

Translation into SPARQL Algebra

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

1. Expand abbreviated IRIs

Translation into SPARQL Algebra

```
{ ?book <http://ex.org/price> ?price
  OPTIONAL { ?book <http://ex.org/title> ?title }
  { ?book <http://ex.org/author>
                  <http://ex.org/Shakespeare> } UNION
  { ?book <http://ex.org/author>
                  <http://ex.org/Marlowe> }
  FILTER (?price < 15)
}
```

Translation into SPARQL Algebra

```
{ ?book <http://ex.org/price> ?price
  OPTIONAL { ?book <http://ex.org/title> ?title }
  { ?book <http://ex.org/author>
                  <http://ex.org/Shakespeare> } UNION
  { ?book <http://ex.org/author>
                  <http://ex.org/Marlowe> }
  FILTER (?price < 15)
}
```

2. Replace triple patterns with operator Bgp(·)

Translation into SPARQL Algebra

```
{ Bgp(?book <http://ex.org/price> ?price)
  OPTIONAL {Bgp(?book <http://ex.org/title> ?title)}
  {Bgp(?book <http://ex.org/author>
                  <http://ex.org/Shakespeare>)} UNION
  {Bgp(?book <http://ex.org/author>
                  <http://ex.org/Marlowe>)}
  FILTER (?price < 15)
}
```

Translation into SPARQL Algebra

```
{ Bgp(?book <http://ex.org/price> ?price)
  OPTIONAL {Bgp(?book <http://ex.org/title> ?title)}
  {Bgp(?book <http://ex.org/author>
                  <http://ex.org/Shakespeare>)} UNION
  {Bgp(?book <http://ex.org/author>
                  <http://ex.org/Marlowe>)}
  FILTER (?price < 15)
}
```

3. Introduce the LeftJoin(·) operator for optional parts

Translation into SPARQL Algebra

```
{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),
           Bgp(?book <http://ex.org/title> ?title),
           true)
  {Bgp(?book <http://ex.org/author>
                 <http://ex.org/Shakespeare>)} UNION
  {Bgp(?book <http://ex.org/author>
                 <http://ex.org/Marlowe>)}
  FILTER (?price < 15)
}
```

Translation into SPARQL Algebra

```
{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),
           Bgp(?book <http://ex.org/title> ?title),
           true)
  {Bgp(?book <http://ex.org/author>
                  <http://ex.org/Shakespeare>)} UNION
  {Bgp(?book <http://ex.org/author>
                  <http://ex.org/Marlowe>)}
  FILTER (?price < 15)
}
```

4. Combine alternative graph patterns with Union(·) operator

⇝ Refers to neighbouring patterns and has higher precedence than conjunction (left associative)

Translation into SPARQL Algebra

```
{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),
           Bgp(?book <http://ex.org/title> ?title),
           true)
  Union(Bgp(?book <http://ex.org/author>
                      <http://ex.org/Shakespeare>),
        Bgp(?book <http://ex.org/author>
                      <http://ex.org/Marlowe>))
  FILTER (?price < 15)
}
```

Translation into SPARQL Algebra

```
{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),
           Bgp(?book <http://ex.org/title> ?title),
           true)
  Union(Bgp(?book <http://ex.org/author>
                      <http://ex.org/Shakespeare>),
        Bgp(?book <http://ex.org/author>
                      <http://ex.org/Marlowe>))
  FILTER (?price < 15)
}
```

5. Apply Join(·) operator to join non-filter elements

Translation into SPARQL Algebra

```
{ Join(
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),
             Bgp(?book <http://ex.org/title> ?title),
             true),
    Union(Bgp(?book <http://ex.org/author>
                    <http://ex.org/Shakespeare>),
          Bgp(?book <http://ex.org/author>
                    <http://ex.org/Marlowe>)))
    FILTER (?price < 15)
}
```

Translation into SPARQL Algebra

```
{ Join(
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),
             Bgp(?book <http://ex.org/title> ?title),
             true),
    Union(Bgp(?book <http://ex.org/author>
                    <http://ex.org/Shakespeare>),
          Bgp(?book <http://ex.org/author>
                    <http://ex.org/Marlowe>)))
    FILTER (?price < 15)
}
```

6. Translate a group with filters with the Filter(·) operator

Translation into SPARQL Algebra

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

Translation into SPARQL Algebra

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

- Online translation tool:
  `http://sparql.org/query-validator.html`

Agenda

1. Recap
2. Evaluation of the SPARQL Algebra
3. SPARQL Algebra Transformation
4. Operators for the Modifiers
5. Summary

Semantics of the SPARQL Algebra Operations

Bgp(P)	match/evaluate pattern P
Join(M_1, M_2)	conjunctive join of solutions M_1 and M_2
Union(M_1, M_2)	union of solutions M_1 with M_2
LeftJoin(M_1, M_2, F)	optional join of M_1 with M_2 with filter constraint F (`true` if no filter given)
Filter(F, M)	filter solutions M with constraint F
Z	empty pattern (identity for join)

SPARQL Solutions

Solutions as partial functions

- Domain: variables from the query
- Range: IRIs $\cup$ blank nodes $\cup$ RDF literals
- Assignment σ for blank nodes in the query
- Evaluation $[\![\mathrm{Bgp}(P)]\!]_{\mathrm{G}}$ of a BGP P over a graph G results in a multi set

Union of Solutions

Definition (Compatibility & Union)

Two solutions μ_1 and μ_2 are compatible if
$\mu_1(x) = \mu_2(x)$ for all x, for which μ_1 and μ_2 are defined.
The union of two compatible solutions μ_1 and μ_2 is defined as:

$$(\mu_1 \cup \mu_2)(x) = \begin{cases} \mu_1(x) & \text{if } x \in \mathrm{dom}(\mu_1) \\ \mu_2(x) & \text{otherwise} \end{cases}$$

$\leadsto$ simple intuition: union of matching table rows

Union of Solutions

Definition (Compatibility & Union)

Two solutions μ_1 and μ_2 are compatible if $\mu_1(x) = \mu_2(x)$ for all x, for which μ_1 and μ_2 are defined.
The union of two compatible solutions μ_1 and μ_2 is defined as:

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

⇝ simple intuition: union of matching table rows

- We now also define the evaluation of the other SPARQL algebra operators

Evaluation of Join$(\cdot)$

For the evaluation of Join(A_1, A_2) over a graph G with A_1, A_2 algebra objects, we define:

- Let $M_1 = [\![A_1]\!]_{\mathsf{G}}$
- Let $M_2 = [\![A_2]\!]_{\mathsf{G}}$
- Let $J(\mu) = \big\{(\mu_1, \mu_2) \mid M_1(\mu_1) > 0, M_2(\mu_2) > 0,$
  μ_1 and μ_2 are compatible and $\mu = \mu_1 \cup \mu_2\big\}$

$\rightsquigarrow$ J defines compatible pairs of solutions from M_1 and M_2

The evaluation $[\![$Join$(A_1, A_2)]\!]_{\mathsf{G}}$ results in

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

Example to Join(·)

We consider Join(A_1, A_2) over a graph G with $[\![A_1]\!]_{\mathsf{G}} = M_1$, $[\![A_2]\!]_{\mathsf{G}} = M_2$ and:

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

Evaluation of Union($\cdot$)

The evaluation of Union(A_1, A_2) over a graph G, written $[\![\text{Union}(A_1, A_2)]\!]_{\mathsf{G}}$, with A_1, A_2 algebra objects results in:

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

Evaluation of Filter($\cdot$)

The evaluation of Filter(F, A) over a graph G, written $[\![\text{Filter}(F, A)]\!]_{\mathsf{G}}$, with F a filter condition and A an algebra object results in:

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

$[\![\mu(F)]\!]$ is the Boolean result of evaluating the filter condition

Evaluation of LeftJoin(·)

The evaluation of LeftJoin(A_1, A_2, F) over a graph G with F a filter condition and A_1, A_2 algebra objects is defined as:

- $M_1 = [\![A_1]\!]_{\mathsf{G}}$
- $M_2 = [\![A_2]\!]_{\mathsf{G}}$

The evaluation $[\![\text{LeftJoin}(A_1, A_2, F)]\!]_{\mathsf{G}}$ of LeftJoin(A_1, A_2, F) over G results in

$$\begin{aligned} &[\![\text{Filter}(F, \text{Join}(A_1, A_2))]\!]_{\mathsf{G}} \cup \\ &\Big\{ (\mu_1, M_1(\mu_1)) \mid \text{for all } \mu_2 \text{ with } M_2(\mu_2) > 0 : \mu_1 \text{ and } \mu_2 \text{ are} \\ &\qquad\qquad \text{incompatible or } [\![(\mu_1 \cup \mu_2)(F)]\!] = \texttt{false} \Big\} \end{aligned}$$

Example

```
@prefix ex:  <http://eg.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:title  "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:title   ?title .  }
  { ?book   ex:author   ex:Shakespeare .  } UNION
  { ?book   ex:author   ex:Marlowe .  }
}
```

```
@prefix ex:  <http://eg.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:title  "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/title> ?title), true),
      Union(Bgp(?book <http://eg.org/author>
                  <http://eg.org/Shakespeare>),
            Bgp(?book <http://eg.org/author>
                  <http://eg.org/Marlowe>))))
```

Example Evaluation (1)

```
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>))))
```

book
ex:Tamburlaine
ex:DoctorFaustus

Example Evaluation (1)

```
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>))))
```

book
ex:Tamburlaine
ex:DoctorFaustus

book
ex:Macbeth
ex:Hamlet

Example Evaluation (2)

```
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>))))
```

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

Example Evaluation (3)

```
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>))))
```

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

Example Evaluation (3)

```
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>))))
```

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

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

Example Evaluation (4)

```
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>))))
```

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

Example Evaluation (5)

```
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>))))
```

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

Example Evaluation (6)

```
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>))))
```

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

Agenda

1. Recap
2. Evaluation of the SPARQL Algebra
3. SPARQL Algebra Transformation
4. Operators for the Modifiers
5. Summary

Formal Algebra Transformation

- During parsing of a query, a parse tree is constructed
- The parse tree contains objects that correspond to the grammar
- For the transformation, we traverse the parse tree and recursively build the algebra objects
- The query pattern is a `GroupGraphPattern` consisting of the following elements:
  - TriplesBlock
  - Filter
  - OptionalGraphPattern
  - GroupOrUnionGraphPattern
  - GraphGraphPattern

Part of the SPARQL Grammar

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

Transformation of `GroupOrUnionGraphPattern`

translate(`GroupOrUnionGraphPattern` G)

Input: a `GroupOrUnionGraphPattern` G
with elements $e_1, \ldots, e_n$

Output: a SPARQL algebra expression A

```
for $i = 1, \ldots, n$ do
    if A is undefined then
        A := translate($e_i$)
    else
        A := Union(A, translate($e_i$))
return $A$
```

Transformation of GraphGraphPattern

translate(GraphGraphPattern G)

```
Input: a GraphGraphPattern
Output: a SPARQL algebra expression A
if G GRAPH IRI GroupGraphPattern then
    A := Graph(IRI, translate(GroupGraphPattern))
else if G GRAPH Var GroupGraphPattern then
    A := Graph(Var, translate(GroupGraphPattern))
return A
```

Transformation of `GroupGraphPattern`

translate(`GroupGraphPattern` G)

Input: a `GroupGraphPattern` G = $(e_1, \ldots, e_n)$
Output: a SPARQL algebra expression A

```
A := Z { the empty pattern}
F := ∅ { filter}
for $i = 1, \ldots, n$ do
    if $e_i$ is FILTER( f ) then
        F := F ∪ $\{f\}$
    else if $e_i$ is OPTIONAL { P } then
        if translate(P) is Filter(F′, A′) then
            A := LeftJoin(A, A′, F′)
        else
            A := LeftJoin(A, translate(P), true)
    else
        A := Join(A, translate(e_i))
if F $\neq$ ∅ then
    A := Filter($\bigwedge_{f \in F}$ f, A)
return A
```

Simplification of Algebra Objects

- Groups with just one pattern (without filters) result in $\text{Join}(Z, A)$ and can be substituted by A
- The empty pattern is the identity for joins:
  - Replace $\text{Join}(Z, A)$ by A
  - Replace $\text{Join}(A, Z)$ by A

Agenda

1. Recap
2. Evaluation of the SPARQL Algebra
3. SPARQL Algebra Transformation
4. Operators for the Modifiers
5. Summary

Operators for Representing the Modifiers

ToList(M)	Constructs from a multi set a sequence with the same elements and multiplicity (arbitrary order, duplicates not necessarily adjacent)
OrderBy(M, comparators)	sorts the solutions
Distinct(M)	removes the duplicates
Reduced(M)	may remove duplicates
Slice(M, o, l)	cuts the solutions to a list of length l starting from position o
Project(M, vars)	projects out the mentioned variables

Transformation of the Modifiers

Let q be a SPARQL query with pattern P and corresponding algebra object A_P. We construct an algebra object A_q for q as follows:

1. $A_q := \mathsf{ToList}(\mathsf{A_P})$
2. $A_q := \mathsf{OrderBy}(\mathsf{A_q}, (\mathsf{c_1}, \ldots, \mathsf{c_n}))$ if q contains an ORDER BY clause with comparators $c_1, \ldots, c_n$
3. $A_q := \mathsf{Project}(\mathsf{A_q}, \mathsf{vars})$ if the result format is `SELECT` with `vars` the selected variables ($\star$ all variables in scope)
4. $A_q := \mathsf{Distinct}(\mathsf{A_q})$ is the result format is `SELECT` and q contains `DISTINCT`
5. $A_q := \mathsf{Reduced}(\mathsf{A_q})$ if the result format is `SELECT` and q contains `REDUCED`
6. $A_q := \mathsf{Slice}(\mathsf{A_q}, \mathsf{start}, \mathsf{length})$ if the query contains `OFFSET start` or `LIMIT length` where `start` defaults to 0 and `length` defaults to $(|[\![A_q]\!]_\mathsf{G}| - \texttt{start})$

Evaluation of the Modifiers

The algebra objects for the modifiers are recursively evaluated

- Evaluate the algebra expression of the operator
- Apply the operations for the solution modifiers to the obtained solutions

Agenda

1. Recap
2. Evaluation of the SPARQL Algebra
3. SPARQL Algebra Transformation
4. Operators for the Modifiers
5. Summary

Summary

- We learned how to evaluate SPARQL queries
- The query is transformed into an algebra object
- The query basic graph patterns generate solutions
- The other operators combine solutions
- The result format determines how the solutions are presented

Outlook

- Next lecture: SPARQL 1.1 features
- Non-Query parts of the specification (Protocol, Service Descriptions, Update, . . .)
- Then: Entailment Regimes (SPARQL with inferred results)