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

```sparql
  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

```sparql
{  ?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

```sparql
    <http://ex.org/Shakespeare> } UNION
    <http://ex.org/Marlowe> }
  FILTER (?price < 15)
}
```
Translation into SPARQL Algebra

    <http://ex.org/Shakespeare> } UNION 
    <http://ex.org/Marlowe> } } 
FILTER (?price < 15)
}

2. Replace triple patterns with operator Bgp(·)
Translation into SPARQL Algebra

  FILTER (?price < 15)
}

Translation into SPARQL Algebra

  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

```sparql
{ LeftJoin(Bgp(?book <http://ex.org/price> ?price),
            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),  
    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),
            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),
            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),
           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),
             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),
  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

<table>
<thead>
<tr>
<th>Operation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bgp($P$)</td>
<td>match/evaluate pattern $P$</td>
</tr>
<tr>
<td>Join($M_1$, $M_2$)</td>
<td>conjunctive join of solutions $M_1$ and $M_2$</td>
</tr>
<tr>
<td>Union($M_1$, $M_2$)</td>
<td>union of solutions $M_1$ with $M_2$</td>
</tr>
<tr>
<td>LeftJoin($M_1$, $M_2$, $F$)</td>
<td>optional join of $M_1$ with $M_2$ with filter constraint $F$ ($true$ if no filter given)</td>
</tr>
<tr>
<td>Filter($F$, $M$)</td>
<td>filter solutions $M$ with constraint $F$</td>
</tr>
<tr>
<td>$Z$</td>
<td>empty pattern (identity for join)</td>
</tr>
</tbody>
</table>
Solutions as partial functions

- Domain: variables from the query
- Range: IRIs $\cup$ blank nodes $\cup$ RDF literals
- Assignment $\sigma$ for blank nodes in the query
- Evaluation $\llbracket Bgp(P) \rrbracket_G$ of a BGP $P$ over a graph $G$ results in a multi set
Union of Solutions

**Definition (Compatibility & Union)**

Two solutions $\mu_1$ and $\mu_2$ are compatible if $\mu_1(x) = \mu_2(x)$ for all $x$, for which $\mu_1$ and $\mu_2$ are defined.

The union of two compatible solutions $\mu_1$ and $\mu_2$ 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
Union of Solutions

Definition (Compatibility & Union)

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

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

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

- We now also define the evaluation of the other SPARQL algebra operators
Evaluation of Join(·)

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

- Let $M_1 = [A_1]_G$
- Let $M_2 = [A_2]_G$
- Let $J(\mu) = \{ (\mu_1, \mu_2) | M_1(\mu_1) > 0, M_2(\mu_2) > 0, \mu_1 \text{ and } \mu_2 \text{ are compatible and } \mu = \mu_1 \cup \mu_2 \}$

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

The evaluation $[[\text{Join}(A_1, A_2)]]_G$ results in

$$\left\{ (\mu, n) | n = \sum_{(\mu_1, \mu_2) \in J(\mu)} (M_1(\mu_1) \ast M_2(\mu_2)) > 0 \right\}$$
Example to Join(·)

We consider Join($A_1, A_2$) over a graph $G$ with $[A_1]_G = M_1$, $[A_2]_G = M_2$ and:

\[
M_1 = \{(\mu_1 : ?x \mapsto ex : a, ?y \mapsto ex : b), 2), \\\n(\mu_2 : ?x \mapsto ex : a, 1)\}
\]

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

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

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

\[
\text{Join}(M_1, M_2) = \left\{(\mu, n) \mid n = \sum_{(\mu_1, \mu_2) \in J(\mu)} (M_1(\mu_1) \ast M_2(\mu_2)) > 0 \right\}
\]

\[
= \{(\mu, 9)\}
\]

\[
n = 2 \ast 3 + 1 \ast 3 = 6 + 3 = 9
\]
Evaluation of Union(·)

The evaluation of $\text{Union}(A_1, A_2)$ over a graph $G$, written $\llbracket \text{Union}(A_1, A_2) \rrbracket_G$, with $A_1, A_2$ algebra objects results in:

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

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

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

$\llbracket \mu(F) \rrbracket$ is the Boolean result of evaluating the filter condition
Evaluation of $\text{LeftJoin}(\cdot)$

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

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

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

$$[\text{Filter}(F, \text{Join}(A_1, A_2))]_G \cup \left\{ (\mu_1, M_1(\mu_1)) \mid \text{for all } \mu_2 \text{ with } M_2(\mu_2) > 0 : \mu_1 \text{ and } \mu_2 \text{ are incompatible or } [((\mu_1 \cup \mu_2)(F)) = \text{false}] \right\}$$
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:author ex:Shakespeare . } UNION 
  { ?book ex:author ex:Marlowe . } 
}

TU Dresden, May 9  Foundations of Semantic Web Technologies  slide 33 of 56
Example Evaluation (1)

Filter(\(\text{\texttt{price}} < 15\),
    Join(
        LeftJoin(Bgp(\(\text{\texttt{book}} < \text{http://eg.org/price} > \text{\texttt{price}}\)),
                 Bgp(\(\text{\texttt{book}} < \text{http://eg.org/title} > \text{\texttt{title}}\)),
                 true),
            Union(Bgp(\(\text{\texttt{book}} < \text{http://eg.org/author} > \text{http://eg.org/Shakespeare}\)),
                    Bgp(\(\text{\texttt{book}} < \text{http://eg.org/author} > \text{http://eg.org/Marlowe}\))))

<table>
<thead>
<tr>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Tamburlaine</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
</tr>
</tbody>
</table>
Example Evaluation (1)

\[
\text{Filter}(\text{?price} < 15, \\
\text{Join(} \\
\quad \text{LeftJoin(Bgp(}\text{?book <http://eg.org/price> ?price)}, \\
\quad \text{Bgp(}\text{?book <http://eg.org/title> ?title})), \\
\quad \text{true}), \\
\text{Union(Bgp(}\text{?book <http://eg.org/author> <http://eg.org/Shakespeare}\rangle, \\
\quad \text{Bgp(}\text{?book <http://eg.org/author> <http://eg.org/Marlowe}\rangle)))}
\]

<table>
<thead>
<tr>
<th>book</th>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Tamburlaine</td>
<td>ex:Macbeth</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>ex:Hamlet</td>
</tr>
</tbody>
</table>
Example Evaluation (2)

Filter(\(\text{price} < 15\),
  Join(
    LeftJoin(Bgp(\(\text{book} < \text{http://eg.org/price} > \text{price}\)),
      Bgp(\(\text{book} < \text{http://eg.org/title} > \text{title}\)),
    true),
  Union(Bgp(\(\text{book} < \text{http://eg.org/author} > \text{http://eg.org/Shakespeare}\)),
    Bgp(\(\text{book} < \text{http://eg.org/author} > \text{http://eg.org/Marlowe}\))))

<table>
<thead>
<tr>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
</tr>
<tr>
<td>ex:Macbeth</td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
</tr>
</tbody>
</table>
Example Evaluation (3)

\[
\text{Filter}(\text{?price } < 15, \\
\text{Join(} \\
\text{  LeftJoin(Bgp(\text{?book } <\text{http://eg.org/price}> \text{?price}),} \\
\text{  Bgp(\text{?book } <\text{http://eg.org/title}> \text{?title}),} \\
\text{  true),} \\
\text{Union(Bgp(\text{?book } <\text{http://eg.org/author}>} \\
\text{    <http://eg.org/Shakespeare>),} \\
\text{  Bgp(\text{?book } <\text{http://eg.org/author}>} \\
\text{    <http://eg.org/Marlowe>))))}
\]

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
</tr>
<tr>
<td>ex:RomeoJulia</td>
<td>9</td>
</tr>
</tbody>
</table>
Example Evaluation (3)

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

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
</tr>
<tr>
<td>ex:RomeoJulia</td>
<td>9</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>book</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:DoctorFaustus</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
</tbody>
</table>
Example Evaluation (4)

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

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
<td></td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
<tr>
<td>ex:RomeoJulia</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
Example Evaluation (5)

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

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
<td>ex:Hamlet</td>
</tr>
<tr>
<td>ex:Tamburlaine</td>
<td>17</td>
<td>ex:Tamburlaine</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
</tbody>
</table>
Example Evaluation (6)

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

<table>
<thead>
<tr>
<th>book</th>
<th>price</th>
<th>title</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Hamlet</td>
<td>10.5</td>
<td></td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
<td>&quot;The Tragical History of Doctor Faustus&quot;</td>
</tr>
</tbody>
</table>
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

\[
\begin{align*}
\text{GroupGraphPattern} & ::= \{'\text{TriplesBlock}\?\text{(}((\text{GraphPatternNotTriples} \mid \text{Filter})\text{)\?'?\text{TriplesBlock}\?\text{)}\)}\text{)*',} \\
\text{GraphPatternNotTriples} & ::= \text{OptionalGraphPattern} \mid \text{GroupOrUnionGraphPattern} \mid \text{GraphGraphPattern} \\
\text{OptionalGraphPattern} & ::= \text{'OPTIONAL'} \text{GroupGraphPattern} \\
\text{GroupOrUnionGraphPattern} & ::= \text{GroupGraphPattern} (\text{'UNION'} \text{GroupGraphPattern})* \\
\text{Filter} & ::= \text{'FILTER'} \text{Constraint}
\end{align*}
\]
Transformation of

`GroupOrUnionGraphPattern`

\[\text{translate}(\text{GroupOrUnionGraphPattern } G)\]

**Input:** a `GroupOrUnionGraphPattern G` with elements \(e_1, \ldots, e_n\)

**Output:** a SPARQL algebra expression \(A\)

1. for \(i = 1, \ldots, n\) do
2. if \(A\) is undefined then
3. \(A := \text{translate}(e_i)\)
4. else
5. \(A := \text{Union}(A, \text{translate}(e_i))\)
6. return \(A\)
Transformation of `GraphPattern`

```plaintext
translate(GraphPattern G)

Input: a `GraphPattern`
Output: a SPARQL algebra expression A

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

\textbf{translate}(\text{GroupGraphPattern } G)

\begin{itemize}
\item \textbf{Input:} a GroupGraphPattern $G = (e_1, \ldots, e_n)$
\item \textbf{Output:} a SPARQL algebra expression $A$
\item $1: A := Z \{ \text{the empty pattern} \}$
\item $2: F := \emptyset \{ \text{filter} \}$
\item $3: \text{for } i = 1, \ldots, n \text{ do}$
\item $4: \quad \text{if } e_i \text{ is } \text{FILTER}(f) \text{ then}$
\item $5: \quad \quad F := F \cup \{f\}$
\item $6: \quad \text{else if } e_i \text{ is } \text{OPTIONAL} \{ P \} \text{ then}$
\item $7: \quad \quad \text{if } \text{translate}(P) \text{ is Filter}(F', A') \text{ then}$
\item $8: \quad \quad \quad A := \text{LeftJoin}(A, A', F')$
\item $9: \quad \quad \text{else}$
\item $10: \quad \quad \quad A := \text{LeftJoin}(A, \text{translate}(P), \text{true})$
\item $11: \quad \text{else}$
\item $12: \quad \quad A := \text{Join}(A, \text{translate}(e_i))$
\item $13: \quad \text{if } F \neq \emptyset \text{ then}$
\item $14: \quad \quad A := \text{Filter}(\land_{f \in F} f, A)$
\item $15: \quad \text{return } A$
\end{itemize}
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

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ToList($M$)</td>
<td>Constructs from a multi set a sequence with the same elements and multiplicity (arbitrary order, duplicates not necessarily adjacent)</td>
</tr>
<tr>
<td>OrderBy($M$, comparators)</td>
<td>sorts the solutions</td>
</tr>
<tr>
<td>Distinct($M$)</td>
<td>removes the duplicates</td>
</tr>
<tr>
<td>Reduced($M$)</td>
<td>may remove duplicates</td>
</tr>
<tr>
<td>Slice($M$, $o$, $l$)</td>
<td>cuts the solutions to a list of length $l$ starting from position $o$</td>
</tr>
<tr>
<td>Project($M$, vars)</td>
<td>projects out the mentioned variables</td>
</tr>
</tbody>
</table>
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. \( A_q := \text{ToList}(A_P) \)
2. \( A_q := \text{OrderBy}(A_q, (c_1, \ldots, c_n)) \) if \( q \) contains an ORDER BY clause with comparators \( c_1, \ldots, c_n \)
3. \( A_q := \text{Project}(A_q, \text{vars}) \) if the result format is \text{SELECT} with \text{vars} the selected variables (\(* \) all variables in scope)
4. \( A_q := \text{Distinct}(A_q) \) is the result format is \text{SELECT} and \( q \) contains \text{DISTINCT}
5. \( A_q := \text{Reduced}(A_q) \) if the result format is \text{SELECT} and \( q \) contains \text{REDUCED}
6. \( A_q := \text{Slice}(A_q, \text{start}, \text{length}) \) if the query contains \text{OFFSET} \text{start} or \text{LIMIT} \text{length} \text{where} \text{start} \text{defaults to 0 and length defaults to} (|\|A_q\|_G| - \text{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)