Semantics of SPARQL

Werner Nutt
Acknowledgment

These slides are essentially identical with those by Sebastian Rudolph for his course on Semantic Web Technologies at TU Dresden
• Output Formats
• SPARQL Semantics
• Transformation of Queries into Algebra Expressions
• Evaluation of the SPARQL Algebra
• Output Formats
• SPARQL Semantics
• Transformation of Queries into Algebra Expressions
• Evaluation of the SPARQL Algebra
Output Format SELECT

So far all results have been tables (solution sequences):
Output format SELECT

Syntax:
• SELECT <VariableList>
• SELECT *

Advantage
• Simple sequential processing of the results

Disadvantage
• Structure/relationships is lost between the expressions in the result
Output Format **CONSTRUCT**

CONSTRUCT creates an RDF graph for the results

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

**Advantage**
- Structured result data with relationships between the elements

**Disadvantage**
- Sequential processing of the results is harder
- No treatment of unbound variables (triples are omitted)
CONSTRUCT Templates with Blank Nodes

@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:surname  ?fname }
CONSTRUCT Templates with Blank Nodes

Resulting RDF Graph

```sparql
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
_:v1 vcard:N _:x1 .
_:x1 vcard:givenName "Alice" ;
vcard:familyName "Hacker" .
_:v2 vcard:N _:x2 .
_:x2 vcard:givenName "Bob" ;
vcard:familyName "Hacker" .
```
**Further Output Formats: ASK & DESCRIBE**

SPARQL supports two additional output formats:

- **ASK** only checks whether the query has at least one answer (true/false result)
- **DESCRIBE** (informative) returns an RDF description for each resulting URI (application dependent)

---

**Sample Query over DBpedia**

```sparql
PREFIX dbo:<http://dbpedia.org/ontology/>
PREFIX dbp:<http://dbpedia.org/property/>

DESCRIBE ?p WHERE {
  ?p a dbo:Person .
  FILTER(REGEX(STR(?n),"ital","i"))
}
```
Output (just the beginning …)

@prefix dbpprop: <http://dbpedia.org/property/> .
@prefix dbpedia: <http://dbpedia.org/resource/> .

dbpedia:Emanuela_Da_Ros dbpprop:birthPlace dbpedia:Italy ;
    dbpprop:placeOfBirth dbpedia:Italy .

@prefix dbpedia-owl: <http://dbpedia.org/ontology/> .

dbpedia:Emanuela_Da_Ros dbpedia-owl:birthPlace dbpedia:Italy .

The answer depends on the implementation and is not defined by the standard
• Output Formats
• SPARQL Semantics
• Transformation of Queries into Algebra Expressions
• Evaluation of the SPARQL Algebra
Semantics of Query Languages

So far only informal presentation of SPARQL features

- User: “Which answers can I expect for my query?”
- Developer: “Which behaviour is expected from my SPARQL implementation?”
- Marketing: “Is our product already conformant with the SPARQL standard?”

⇒ Formal semantics should clarify these questions …
Logic-based Semantics

Semantics of formal logics:

• **Model-theoretic semantics**: Which interpretations do satisfy my given formulas?

• **Proof-theoretic semantics**: Which new formulas can be derived from my given formulas?

• ...


Semantics of Programming Languages

- **Axiomatic semantics:**
  Which logical statements hold for my program?

- **Operational semantics:**
  What happens during the processing of my program?

- **Denotational semantics:**
  How can we describe the input/output function of the program in an abstract way?
Semantics of Query Languages (1)

Query Entailment

Logical view of queries and databases
  – Query as description of allowed results
  – Data as set of logical assumptions (axiom set/theory)
  – Results as logical entailment

• OWL DL and RDF(S) as query languages

• Also logic programming view …
Semantics of Query Languages (2)

Query Algebra

Query as instruction for computing the results
• Queried data as input
• Results as output

⇒ Relational algebra for SQL
⇒ SPARQL algebra

See http://www.w3.org/2001/sw/DataAccess/rq23/rq24-algebra
• Output Formats
• SPARQL Semantics
• Transformation of Queries into Algebra Expressions
• Evaluation of the SPARQL Algebra
Translation into SPARQL Algebra

  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
  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

```
  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

② Replace triple patterns (= basic graph patterns) with operator $\text{Bgp}(\cdot)$
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)  
}
```
Translation into SPARQL Algebra

```sparql
  {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),
           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)
}
```

③ Introduce the LeftJoin(·) operator for optional parts

Note: LeftJoin(·, ·, ·) is a ternary operator

- 1\textsuperscript{st} argument: mandatory part
- 2\textsuperscript{nd} argument: Bgps of optional part
- 3\textsuperscript{rd} argument: the filters of the optional group
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 the Union(·, ·) operator
⇒ Refers to neighbouring patterns and has higher precedence than conjunction (left associative)
Translation into SPARQL Algebra

```sparql
{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

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

⑤ Apply \text{Join(·;·)} operator to join non-filter elements
Translation into SPARQL Algebra

```sparql
{Join(
  LeftJoin(Bgp(?book <http://ex.org/price> ?price),
    true),
  Union(Bgp(?book <http://ex.org/author> http://ex.org/Shakespeare),
FILTER (?price < 15)
}
```
Translation into SPARQL Algebra

```sparql
{Join(
    LeftJoin(Bgp(?book <http://ex.org/price> ?price),
             true),
    Union(Bgp(?book <http://ex.org/author> http://ex.org/Shakespeare),
  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),

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),

Online translation tool:

http://sparql.org/query-validator.html
• Output Formats
• SPARQL Semantics
• Transformation of Queries into Algebra Expressions
• Evaluation of the SPARQL Algebra
Semantics of the SPARQL Algebra Operations

Now we have an algebra expression, but what do the algebra operations mean?

<table>
<thead>
<tr>
<th>Algebra Operator</th>
<th>Intuitive Semantics</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>

Only Bgp($\cdot$) matches or evaluates graph fragments …
Semantics of the SPARQL Algebra Operations

How can we define that more formally?

Output:
• “solution set” (formatting irrelevant)

Input:
• Queried (active) graph
• Partial results from previous evaluation steps
• Different parameters according to the operation

➤ How can we formally describe the “results”?
SPARQL Results

Intuition:
• Results are as for relational queries: tables of variable assignments

Result:
  List of solutions (solution sequence)
  
  ➔ each solution corresponds to one table row
SPARQL Results

Formally:

A solution is a partial function (also called “mapping”) with
• **Domain**: relevant variables
• **Range**: IRIs ∪ blank nodes ∪ RDF literals

⇒ Unbound variables are those that have no assigned value (partial function)
⇒ Mappings are denoted by the greek letter $\mu$
Evaluation of Basic Graph Patterns

Definition (Solution of a BGP)

Let \( P \) be a basic graph pattern.

A partial function/mapping \( \mu \) is a solution for \( \text{Bgp}(P) \) over the queried (active) graph \( G \) if:

1. the domain of \( \mu \) is exactly the set of variables in \( P \),
2. there exists an assignment \( \sigma \) from blank nodes in \( P \) to IRIs, blank nodes, or RDF literals such that:
3. the RDF graph \( \mu(\sigma(P)) \) is a subgraph of \( G \)
Remarks on the Definition

• If there were only variables, we would only talk about $\mu$.
• Since also the blank nodes need to be interpreted, there is also $\sigma$.
• It is first $\sigma$ and then $\mu$ because we want
  – that $\sigma$ only binds blank nodes in $P$,
  – not the blank nodes introduced by $\mu$.
• The result of evaluating $\text{Bgp}(P)$ over $G$ is written

$$[[\text{Bgp}(P)]]_G$$

• The result is a multiset of solutions $\mu$.
• The multiplicity of each solution $\mu$ corresponds to the number of different assignments $\sigma$
Multisets

Definition (Multi Set)

• A multiset over a set $S$ is a function $M$ that assigns to every element $s$ of $S$
  – a natural number $M(s)$ such that $M(s) \geq 0$ or $M(s) = \infty$ (infinity)
• $M(s)$ is the multiplicity of $s$ in $M$.

Alternative notation: $\{\{ a, b, b \}\}$ corresponds to the multiset $M$ over $\{a, b, c\}$ with $M(a) = 1$, $M(b) = 2$, and $M(c) = 0$. 
Solution Mappings: Example

```
ex:Werner ex:give [ 
    a ex:Lecture ; 
    ex:hasTopic "SPARQL" ] .
ex:Fariz ex:give [ 
    a ex:Lab ; 
    ex:hasTopic "Jena" ] .
```

Bgp( ?who ex:give _:_x . _:_x ex:hasTopic ?what)

Question:
• What are the $\sigma$s and the $\mu$s?
• What are the solutions? And what is their multiplicity?

*Hint: As a first step, write the data as a set of triples.*
Solution Mappings: Exercise

ex:Fariz ex:gives [ 
  a ex:Lab ; 
  ex:hasTopic "RDF" ] .
ex:Fariz ex:gives [ 
  a ex:Lab ; 
  ex:hasTopic "Jena" ] .

Bgp( ?who ex:gives _:x . _:x ex:hasTopic ?what)

Question:
• What are the $\sigma$s and the $\mu$s?
• What are the solutions? And what is their multiplicity?

*Hint: As a first step, write the data as a set of triples.*
Union of Solutions (1)

Definition (Compatibility)

Two solutions $\mu_1$ and $\mu_2$ are compatible if

$\mu_1(x) = \mu_2(x)$ for all variables $x$,

for which $\mu_1$ and $\mu_2$ are defined

Exercise: Find examples of $\mu_1$ and $\mu_2$ that are compatible/not compatible
Union of Solutions (2)

Definition (Union)

The union of two compatible solutions $\mu_1$ and $\mu_2$ is a/the $\mu$ such that

- $\mu(x) = \mu_1(x)$ if $x$ is in $\text{dom}(\mu_1)$
- $\mu(x) = \mu_2(x)$ if $x$ is in $\text{dom}(\mu_2)$

Where does the compatibility play a role?
Evaluation of \( \text{Join}(\cdot,\cdot) \) (1)

To define the evaluation of a join expression \( \text{Join}(E_1,E_2) \) over a graph \( G \) we proceed in two steps:

1. We define the join \( \text{Join}(M_1,M_2) \) of two multisets of mappings.

2. We define the evaluation \( [[[\text{Join}(E_1,E_2)]]_G \) of a join expression as the join of the evaluations \( [[[E_1]]_G \) and \( [[[E_2]]_G \) of the arguments.
Evaluation of Join(\cdot,\cdot)  \ (2)

For a mapping $\mu$ and multisets of mappings $M_1, M_2$ we define the set of join combinations of $\mu$ as

$$J(\mu) = \{ (\mu_1, \mu_2) \mid M_1(\mu_1) > 0, M_2(\mu_2) > 0, \mu_1 \text{ and } \mu_2 \text{ are compatible and } \mu = \mu_1 \cup \mu_2 \}$$

That is, $J(\mu)$ consists of all possible ways to obtain $\mu$ as a combination of mappings in $M_1, M_2$
Evaluation of Join(\cdot,\cdot)  (3)

For multisets of mappings $M_1, M_2$ we define

$$
\text{Join}(M_1, M_2) := \{ (\mu, n) | n = \sum_{(\mu_1, \mu_2) \in \text{j}(\mu)} (M_1(\mu_1) \ast M_2(\mu_2)) \}
$$

That is,

- \text{Join}(M_1, M_2) consists of all mappings $\mu$ that can be combined out of mappings in $M_1$ and $M_2$
- If $\mu$ can be combined out of $\mu_1$ and $\mu_2$, and $\mu_1$ occurs $n_1$ times in $M_1$ and $\mu_2$ occurs $n_2$ times in $M_2$, then this combination contributes $n_1 \ast n_2$ to the multiplicity of $\mu$ in $\text{Join}(M_1, M_2)$
Evaluation of $\text{Join}(\cdot,\cdot)$ (4)

Let $E_1, E_2$ be algebra expressions and let $G$ be a graph.

Then we define

$$[[\text{Join}(E_1,E_2)]_G] := \text{Join}( [[E_1]]_G, [[E_2]]_G )$$

In words: we evaluate the join of $E_1$ and $E_2$ by
- first evaluating $E_1$ and $E_2$ separately
- and then taking the join of the resulting multisets of mappings
Exercise for Join(\cdot,\cdot)

We consider algebra expressions $E_1$, $E_2$ and a graph $G$ such that $[[E_1]]_G = M_1$ and $[[E_2]]_G = M_2$. We want to compute $\text{Join}(E_1,E_2)$ over $G$. Suppose

$$M_1 = \{ (\mu_1: ?x \rightarrow \text{ex:a}, \, ?y \rightarrow \text{ex:b}) ,2),
(\mu_2: \, ?x \rightarrow \text{ex:a}, \, 1) \}$$

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

What is $\text{Join}(M_1,M_2)$?

I.e., which are the elements of the join?

And what is their multiplicity?
Evaluation of Union (1)

We first define the union of two multisets of assignments, and then the evaluation of a union expression.

Let $M_1, M_2$ be multisets of mappings. Then

$$\text{Union}(M_1, M_2) := \{ (\mu, n) \mid n = M_1(\mu) + M_2(\mu) > 0 \}$$

In words:

- the union contains the mappings that occur at least once in $M_1$ or $M_2$
- the multiplicity of a mapping $\mu$ in the union is the sum of the multiplicities in $M_1$ and $M_2$
Evaluation of Union (2)

Let $E_1$, $E_2$ be algebra expressions and let $G$ be a graph.

Then we define

$$[[\text{Union}(E_1,E_2)]_G := \text{Union}( [[E_1]]_G , [[E_1]]_G )$$

In words: we evaluate the union of $E_1$ and $E_2$ by

- first evaluating $E_1$ and $E_2$ separately
- and then taking the union of the resulting multisets of mappings
Evaluation of Filter(\cdot,\cdot) (1)

To define the evaluation of a filter expression $\text{Filter}(F,E)$ over a graph $G$ we proceed in two steps:

① We define the filter operation $\text{Filter}(F,M)$ of a filter condition $F$ and a multiset of mappings $M$

② We define the evaluation $[[\text{Filter}(F,E)]]_G$ of a filter expression as the filter operation by $F$ on the evaluation $[[E]]_G$
Evaluation of Filter(·) (2)

For a filter condition $F$ and multiset of mappings $M$ we define

$$\text{Filter}(F, M) := \{ (\mu, n) \mid M(\mu) = n > 0 \text{ and } \mu(F) = T \}$$

Here, $\mu(F)$ is the truth value (i.e., one of T, E, F) obtained from evaluating $F$ with respect to $\mu$.

The definition says that

1. all mappings survive that satisfy the filter condition, and
2. they survive with the multiplicity they had in $M$
Evaluation of Filter(·) (3)

Let $E$ be an algebra expression, $F$ a filter, and $G$ a graph. The we define

$$[[\text{Filter}(F, E)]]_G := \text{Filter}(F, [[E]]_G)$$

In words,
- we first evaluate $E$ and
- then apply the filter $F$
Evaluation of LeftJoin(·) (1)

Again, to define the evaluation of a left join expression $\text{LeftJoin}(E_1, E_2, F)$ over a graph $G$ we proceed in two steps:

① We define the left join $\text{LeftJoin}(M_1, M_2, F)$ of two multisets of mappings and a filter condition.

② We define the evaluation $[[\text{Join}(E_1, E_2, F)]]_G$ of a left join expression as the left join of the evaluations $[[E_1]]_G$ and $[[E_2]]_G$ of the arguments with respect to $F$. 
Evaluation of LeftJoin(·) (2)

Let $M_1$, $M_2$ be multisets of mappings and let $F$ be a filter expression.

We define

$$\text{LeftJoin}(M_1, M_2, F) :=$$

$$\text{Filter}(F, \text{Join}(M_1, M_2)) \cup$$

$$\{ (\mu_1, M_1(\mu_1)) \mid \text{for all } \mu_2 \text{ with } M_2(\mu_2) > 0: \mu_1 \text{ and } \mu_2 \text{ are incompatible}$$

$$\text{or } (\mu_1 \cup \mu_2)(F) \neq T \}$$

That is

- we join and filter as usual, and
- we keep those mappings from $M_1$ that
  - either do not find a match in $M_2$, or
  - for which none of the combinations with a match satisfies $F$
Evaluation of $\text{LeftJoin}(\cdot)$ (3)

Let $E_1, E_2$ be algebra expressions, $F$ a filter, and $G$ a graph. Then we define

$$[[\text{LeftJoin}(E_1, E_2, F)]]_G$$

$$:= \text{LeftJoin}([[E_1]]_G, [[E_2]]_G, F)$$
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:RomeusJuliet   ex:author   ex:Brooke ;
                 ex:price    "12"^^xsd:integer .

FILTER (?price < 15)
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:RomeusJuliet   ex:author   ex:Brooke ;
ex:price          "12"^^xsd:integer .

Filter(?price < 15,
     Join(LeftJoin(Bgp(?book ex:price ?price),
                    Bgp(?book ex:title ?title), true),
       Union(Bgp(?book ex:author ex:Shakespeare),
            Bgp(?book ex:author ex:Marlowe))))
Example Evaluation

Filter(?price < 15,
    Join(LeftJoin(Bgp(?book ex:price ?price)
        Bgp(?book ex:title ?title), true),
    Union(Bgp(?book ex:author ex:Shakespeare),
        Bgp(?book ex:author ex:Marlowe))))

<table>
<thead>
<tr>
<th>book</th>
</tr>
</thead>
<tbody>
<tr>
<td>ex:Macbeth</td>
</tr>
<tr>
<td>ex:Hamlet</td>
</tr>
</tbody>
</table>
Example Evaluation

Filter(?price < 15,
    Join(LeftJoin(Bgp(?book ex:price ?price)
        Bgp(?book ex:title ?title), true),
    Union(Bgp(?book ex:author ex:Shakespeare),
        Bgp(?book ex:author ex:Marlowe))))

book
ex:Tamburlaine
ex:DoctorFaustus
Example Evaluation

Filter(?price < 15, 
  Join(LeftJoin(Bgp(?book ex:price ?price) 
    Bgp(?book ex:title ?title), true), 
  Union(Bgp(?book ex:author ex:Shakespeare), 
    Bgp(?book ex:author ex:Marlowe)))))

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

Filter(?price < 15,
  Join(LeftJoin(Bgp(?book ex:price ?price)
    Bgp(?book ex:title ?title), true),
  Union(Bgp(?book ex:author ex:Shakespeare),
    Bgp(?book ex:author ex: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:Tamburlain</td>
<td>17</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
</tr>
<tr>
<td>ex:RomeusJuliet</td>
<td>9</td>
</tr>
</tbody>
</table>
Example Evaluation

Filter(?price < 15,
  Join(LeftJoin(Bgp(?book ex:price ?price)
   Bgp(?book ex:title ?title), true),
  Union(Bgp(?book ex:author ex:Shakespeare),
   Bgp(?book ex:author ex: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:Tamburlain</td>
<td>17</td>
</tr>
<tr>
<td>ex:DoctorFaustus</td>
<td>12</td>
</tr>
<tr>
<td>ex:RomeusJuliet</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

Filter(?price < 15, 
  Join(LeftJoin(Bgp(?book ex:price ?price) 
    Bgp(?book ex:title ?title), true), 
  Union(Bgp(?book ex:author ex:Shakespeare), 
    Bgp(?book ex:author ex: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:Tamburlain</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:RomeusJuliet</td>
<td>9</td>
<td></td>
</tr>
</tbody>
</table>
Example Evaluation

Filter(?price < 15, 
  
  Join(LeftJoin(Bgp(?book ex:price ?price) 
    Bgp(?book ex:title ?title), true), 
  Union(Bgp(?book ex:author ex:Shakespeare), 
    Bgp(?book ex:author ex: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:Tamburlain</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>
</tbody>
</table>
Example Evaluation

Filter(?price < 15,
  Join(LeftJoin(Bgp(?book ex:price ?price)
    Bgp(?book ex:title ?title), true),
  Union(Bgp(?book ex:author ex:Shakespeare),
    Bgp(?book ex:author ex: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>
Formal Algebra Transformation

- During parsing of a query, a parse tree is constructed
- The parse tree contains expressions that correspond to the grammar
- For the transformation, we traverse the parse tree and recursively build the algebra expressions
- 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 Algorithm to Algebra

Algorithm 1 translate(G)

Input: a query pattern G
Output: a SPARQL algebra expression A

1: if G is a Triplesblock then
2:   A := Bgp(G)
3: else if G is a GroupOrUnionGraphPattern then
4:   A := trnslGroupOrUnionGP(G)
5: else if G is a GraphGraphPattern then
6:   A := trnslGraphGP(G)
7: else if G is a GroupGraphPattern then
8:   A := trnslGroupGP(G)
9: return A
Transformation of GroupOrUnionGraphPattern

Algorithm 2  \text{trnslGroupOrUnionGP}(G)

\textbf{Input}: a \text{GroupOrUnionGraphPattern} \, G \\
with elements \, e_1, \ldots, e_n \\
\textbf{Output}: a SPARQL algebra expression \, A \\

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

Algorithm 3 \( \text{trnsIGraphGP}(G) \)

**Input:** a GraphGraphPattern \( G \)

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

1: if \( G \) has the form \( \text{GRAPH IRI GroupGraphPattern} \) then
2: \( A := \text{Graph(IRI, translate(\text{GroupGraphPattern})} \)
3: else if \( G \) has the form \( \text{GRAPH Var GroupGraphPattern} \) then
4: \( A := \text{Graph(Var, translate(\text{GroupGraphPattern})} \)
5: return \( A \)
Transformation of GroupGraphPattern

Algorithm 4  \( \text{trnslGroupGP}(G) \)

**Input:** a GroupGraphPattern \( G = (e_1, \ldots, e_n) \)

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

1:  \( A := Z \) // the empty pattern
2:  \( F := \emptyset \) // the empty filter
3:  \textbf{for} \( i = 1, \ldots, n \) \textbf{do}
4:      \textbf{if} \( e_i \) is of the form \( \text{FILTER}(f) \) \textbf{then}
5:          \( F := F \cup \{f\} \)
6:      \textbf{else if} \( e_i \) is of the form \( \text{OPTIONAL} \{P\} \) \textbf{then}
7:          \textbf{if} \( \text{translate}(P) \) is of the form \( \text{Filter}(F', A') \) \textbf{then}
8:              \( A := \text{LeftJoin}(A, A', F') \)
9:      \textbf{else}
10:         \( A := \text{LeftJoin}(A, \text{translate}(P), \text{true}) \)
Transformation of GroupGraphPattern (cntd)

11: \textbf{else}
12: \hspace{1em} A := \text{Join}(A, \text{translate}(e_i))
13: \textbf{if} \ F \neq \emptyset \ \textbf{then}
14: \hspace{1em} A := \text{Filter}(\land_{f \in F} f, A)
15: \textbf{return} \ A
Simplification of Algebra Expressions

• 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 \( A \)
## 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 multiset 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 expression $A_P$. We construct an algebra expression $A_q$ 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. $A_q := \text{Project}(A_q, vars)$ if the result format is SELECT with $vars$ the selected variables (* all variables in scope)
Transformation of the Modifiers

4. \( A_q := \text{Distinct}(A_q) \) if the result format is SELECT and \( q \) contains DISTINCT

5. \( A_q := \text{Reduced}(A_q) \) if the result format is SELECT and \( q \) contains REDUCED

6. \( A_q := \text{Slice}(A_q, \text{start}, \text{length}) \) if the query contains OFFSET \( \text{start} \) or LIMIT \( \text{length} \) where \( \text{start} \) defaults to 0 and \( \text{length} \) defaults to \((\| [A_q] \|_G \| - \text{start})\)
Evaluation of the Modifiers

The algebra expressions for the modifiers are recursively evaluated

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