FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

Semantics of SPARQL

Sebastian Rudolph
The SPARQL Query Language
Agenda

1 Recap
2 Output Formats
3 SPARQL Semantics
4 Transformation of Queries into Algebra Objects
5 Evaluation of the SPARQL Algebra
6 Summary
Agenda

1. Recap
2. Output Formats
3. SPARQL Semantics
4. Transformation of Queries into Algebra Objects
5. Evaluation of the SPARQL Algebra
6. 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**
Agenda

1 Recap
2 Output Formats
3 SPARQL Semantics
4 Transformation of Queries into Algebra Objects
5 Evaluation of the SPARQL Algebra
6 Summary
Output Format SELECT

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

Syntax: SELECT <VariableList> or SELECT *

Advantage
Simple sequential processing of the results

Disadvantage
Structure/relationships between the objects in the results is lost
Output Format **CONSTRUCT**

**CONSTRUCT** creates an RDF graph for the results

**Example Query**

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

**Advantage**

Structured result data with relationships between the elements

**Disadvantages**

• Sequential processing of the results is harder
• No treatment of unbound variables (triples are omitted)
**Output Format** CONSTRUCT

**CONSTRUCT** creates an RDF graph for the results

**Example Query**

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

**Advantage**

Structured result data with relationships between the elements

**Disadvantages**

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

Data

@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:a foaf:firstname "Alice" ;
    foaf:surname "Hacker" .
_:b foaf:firstname "Bob" ;
    foaf:surname "Hacker" .
CONSTRUCT Templates with Blank Nodes

Data

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

Query

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

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

### Example Query

```
DESCRIBE ?x WHERE { ?x <http://ex.org/emplID> "123" }
```

### Possible Result (prefix declarations omitted):

```
_:a exOrg:emplID "123" ;
foaf:mbox_sha1sum "ABCD1234" ;
vcard:N
[ vcard:Family "Smith" ;
  vcard:Given "John" ] .
foaf:mbox_sha1sum a owl:InverseFunctionalProperty .
```
Agenda

1 Recap

2 Output Formats

3 SPARQL Semantics

4 Transformation of Queries into Algebra Objects

5 Evaluation of the SPARQL Algebra

6 Summary
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 knowledge base?
- Proof-theoretic semantics: Which derivations can be build from my knowledge base?
- ...
Logic-based Semantics

Semantics of formal logics:

- Model-theoretic semantics: Which interpretations do satisfy my knowledge base?
- Proof-theoretic semantics: Which derivations can be build from my knowledge base?
- …
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 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?

What to do with query languages?
Semantics of Query Languages (1)

Query Entailment

- 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
⇝ conjunctive queries
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
Agenda

1. Recap
2. Output Formats
3. SPARQL Semantics
4. Transformation of Queries into Algebra Objects
5. Evaluation of the SPARQL Algebra
6. Summary
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

```
{ ?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

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

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

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

\{
{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

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

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)
}
```
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

\{
  \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)
\}
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 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)
}
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)
}
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>),

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

1. Recap
2. Output Formats
3. SPARQL Semantics
4. Transformation of Queries into Algebra Objects
5. Evaluation of the SPARQL Algebra
6. Summary
Semantics of the SPARQL Algebra

Operations

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

<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>
Semantics of the SPARQL Algebra Operations

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

<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(M₁, M₂)</td>
<td>conjunctive join of solutions M₁ and M₂</td>
</tr>
<tr>
<td>Union(M₁, M₂)</td>
<td>union of solutions M₁ with M₂</td>
</tr>
<tr>
<td>LeftJoin(M₁, M₂, F)</td>
<td>optional join of M₁ with M₂ 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(·) matches or evaluates graph patterns

⇝ We can use entailment checking rather than graph matching
Definition of the SPARQL Operators

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 coded as tables of variable assignments

**Result:**

List of solutions (solution sequence)

〜〜 each solution corresponds to one table row
SPARQL Results

Intuition: Results coded as tables of variable assignments

Result:
List of solutions (solution sequence)

⇝ each solution corresponds to one table row

Solution:
Partial function
- Domain: relevant variables
- Range: IRIs \cup blank nodes \cup RDF literals

⇝ Unbound variables are those that have no assigned value (partial function)
Evaluation of Basic Graph Patterns

**Definition (Solution)**

Let $P$ be a basic graph pattern. A partial function $\mu$ 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 $G$.  

Evaluation of Basic Graph Patterns

- The result of evaluating $Bgp(P)$ over $G$ is written $[Bgp(P)]_G$
- The result is a multi set of solutions $\mu$
- The multiplicity of each solution $\mu$ corresponds to the number of different assignments $\sigma$
Multi Sets

Definition (Multi Set)

A multi set over a set $S$ is a total function $M : S \to \mathbb{N}^+ \cup \{\omega\}$

- $\mathbb{N}^+$ denotes the positive natural numbers
- $\omega > n$ for all $n \in \mathbb{N}^+$
- $M(s)$ is the multiplicity of $s \in S$
- $\omega$: countably infinite number of occurrences

- We represent a multi set over the set $S$ also with the set $\{(s, M(s)) \mid s \in S\}$
- We write $(s, n) \in M$ if $M(s) = n$
- We assume that $M(s) = 0$ if $s \notin S$
- Alternative notation: \{a, b, b\} corresponds to the multi set $M$ over the set \{a, b\} with $M(a) = 1$ and $M(b) = 2$
Solution Mapping Example

ex:Birte ex:gives [  
a ex:Lecture ;  
ex:hasTopic "SPARQL" ] .
ex:Sebastian ex:gives [  
a ex:Lecture ;  
ex:hasTopic "DLs and OWL" ] .

Bgp(?who ex:gives _:_x .  _:_x ex:hasTopic ?what)
Solution Mapping Example

```
ex:Birte ex:gives _:a .
_:a rdf:type ex:Lecture .
_:a ex:hasTopic "SPARQL" .
ex:Sebastian ex:gives _:b .
_:b rdf:type ex:Lecture .
_:b ex:hasTopic "DLs and OWL" .

Bgp(?who ex:gives _:x . _:x ex:hasTopic ?what)
```
Solution Mapping Example

:_ :a rdf:type ex:Lecture.
:_ :a ex:hasTopic "SPARQL".
:_ :b rdf:type ex:Lecture.
:_ :b ex:hasTopic "DLs and OWL".

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

$\mu_1$: ?who $\mapsto$ ex:Birte, ?what $\mapsto$ "SPARQL"
$\sigma_1$: _ :x $\mapsto$ _ :a
Solution Mapping Example


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

\[ \mu_1: \quad ?who \leftrightarrow \text{ex:Birte,} \quad ?what \leftrightarrow \text{"SPARQL"} \]
\[ \sigma_1: \quad _ :x \leftrightarrow _ :a \]
\[ \mu_2: \quad ?who \leftrightarrow \text{ex:Sebastian,} \quad ?what \leftrightarrow \text{"DLs and OWL"} \]
\[ \sigma_2: \quad _ :x \leftrightarrow _ :b \]
Solution Mapping Example

ex:Birte ex:gives _:_a .
_:_a rdf:type ex:Lecture .
_:_a ex:hasTopic "SPARQL" .
ex:Sebastian ex:gives _:_b .
_:_b rdf:type ex:Lecture .
_:_b ex:hasTopic "DLs and OWL" .

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

\[ \begin{align*}
\mu_1 : & \quad ?\text{who} & \mapsto & \text{ex:Birte}, \\
& & & ?\text{what} \mapsto "\text{SPARQL}" \\
\sigma_1 : & \quad _:_x & \mapsto & _:_a \\
\mu_2 : & \quad ?\text{who} & \mapsto & \text{ex:Sebastian}, \\
& & & ?\text{what} \mapsto "\text{DLs and OWL}" \\
\sigma_2 : & \quad _:_x & \mapsto & _:_b
\end{align*} \]

Two solutions each with multiplicity 1
Exercise Solution Sets

ex:Birte ex:gives [ a ex:Lecture ; ex:hasTopic "SPARQL" ] .

\[ \text{Bgp}(\text{who ex:gives } _:x . _:x \text{ ex:hasTopic } _:y) \]
Solution
Solution

ex:Birte ex:gives _:a .
_:a rdf:type ex:Lecture .
_:a ex:hasTopic "SPARQL" .
ex:Birte ex:gives _:b .
_:b rdf:type ex:Lecture .
_:b ex:hasTopic "SPARQL Algebra" .

\[ \text{Bgp}(\text{?who ex:gives } _:x . \text{ } _:x \text{ ex:hasTopic } _:y) \]
Solution

```sparql
_ :a rdf:type ex:Lecture .
_ :a ex:hasTopic "SPARQL" .
ex:Birte ex:give _ :b .
_ :b rdf:type ex:Lecture .
_ :b ex:hasTopic "SPARQL Algebra" .
```

\[ \text{Bgp} \left( \text{?who ex:give _ :x . _ :x ex:hasTopic _ :y} \right) \]

\[ \mu_1: \quad \text{?who } \mapsto \text{ex:Birte,} \]
\[ \sigma_1: \quad _ :x \mapsto _ :a \quad _ :y \mapsto \text{"SPARQL"} \]
Solution

ex:Birte ex:gives _:a .
_:a rdf:type ex:Lecture .
_:a ex:hasTopic "SPARQL" .

ex:Birte ex:gives _:b .
_:b rdf:type ex:Lecture .
_:b ex:hasTopic "SPARQL Algebra" .

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

\[ \mu_1: \quad ?who \mapsto ex: Birte, \]
\[ \sigma_1: \quad _:x \mapsto _:a \quad \text{and} \quad _:y \mapsto "\text{SPARQL}" \]
\[ \mu_2: \quad ?who \mapsto ex: Birte, \]
\[ \sigma_2: \quad _:x \mapsto _:b \quad \text{and} \quad _:y \mapsto "\text{SPARQL Algebra}" \]
Solution

ex:Birte ex:gives _:a .
_:_a rdf:type ex:Lecture .
_:_a ex:hasTopic "SPARQL" .
ex:Birte ex:gives _:b .
_:_b rdf:type ex:Lecture .
_:_b ex:hasTopic "SPARQL Algebra" .

\[ \text{Bgp}(\ ?\text{who} \ ex:\text{gives} \ _:_x . \ _:_x \ ex:\text{hasTopic} \ _:_y) \]

\[ \mu_1: \ ?\text{who} \mapsto \text{ex:\text{Birte}}, \]
\[ \sigma_1: \ _:_x \mapsto \ _:_a \quad \_:_y \mapsto "\text{SPARQL}" \]
\[ \mu_2: \ ?\text{who} \mapsto \text{ex:\text{Birte}}, \]
\[ \sigma_2: \ _:_x \mapsto \ _:_b \quad \_:_y \mapsto "\text{SPARQL Algebra}" \]

One solution with multiplicity 2
Union of Solutions (1)

Definition (Compatibility)

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.
Definition (Compatibility)

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.

\[
\begin{align*}
\mu_1 : \ ?x & \mapsto \text{ex} : a, \ ?y \mapsto \text{ex} : b \\
\mu_2 : \ ?y & \mapsto \text{ex} : b, \ ?z \mapsto \text{ex} : c
\end{align*}
\]
Union of Solutions (1)

Definition (Compatibility)

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.

\[ \mu_1: \ ?x \mapsto \text{ex:} \ a, \ ?y \mapsto \text{ex:} \ b \]
\[ \mu_2: \ ?y \mapsto \text{ex:} \ b, \ ?z \mapsto \text{ex:} \ c \]
Union of Solutions (1)

Definition (Compatibility)

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.

$\mu_1: \ ?x \mapsto \text{ex} : a, \ ?y \mapsto \text{ex} : b$
$\mu_2: \ ?y \mapsto \text{ex} : b, \ ?z \mapsto \text{ex} : c \ \checkmark$

$\mu_1: \ ?x \mapsto \text{ex} : a, \ ?y \mapsto \text{ex} : b$
$\mu_2: \ ?x \mapsto \text{ex} : b, \ ?z \mapsto \text{ex} : c$
Union of Solutions (1)

Definition (Compatibility)

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.

$\mu_1: \text{?x} \mapsto \text{ex: a, ?y} \mapsto \text{ex: b}$
$\mu_2: \text{?y} \mapsto \text{ex: b, ?z} \mapsto \text{ex: c}$  ✓

$\mu_1: \text{?x} \mapsto \text{ex: a, ?y} \mapsto \text{ex: b}$
$\mu_2: \text{?x} \mapsto \text{ex: b, ?z} \mapsto \text{ex: c}$  ❌
Union of Solutions (1)

Definition (Compatibility)

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.

\[
\begin{align*}
\mu_1 &: \ ?x \mapsto \text{ex : a}, \ ?y \mapsto \text{ex : b} \\
\mu_2 &: \ ?y \mapsto \text{ex : b}, \ ?z \mapsto \text{ex : c} \quad \checkmark \\
\mu_1 &: \ ?x \mapsto \text{ex : a}, \ ?y \mapsto \text{ex : b} \\
\mu_2 &: \ ?x \mapsto \text{ex : b}, \ ?z \mapsto \text{ex : c} \quad \notin \\
\mu_1 &: \ ?x \mapsto \text{ex : a} \\
\mu_2 &: \ ?y \mapsto \text{ex : b}
\end{align*}
\]
Union of Solutions (1)

Definition (Compatibility)

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.

\[
\begin{align*}
\mu_1 & : ?x \mapsto \text{ex : a}, \ ?y \mapsto \text{ex : b} \\
\mu_2 & : ?y \mapsto \text{ex : b}, \ ?z \mapsto \text{ex : c} \quad \checkmark
\end{align*}
\]

\[
\begin{align*}
\mu_1 & : ?x \mapsto \text{ex : a}, \ ?y \mapsto \text{ex : b} \\
\mu_2 & : ?x \mapsto \text{ex : b}, \ ?z \mapsto \text{ex : c} \quad \checkmark
\end{align*}
\]

\[
\begin{align*}
\mu_1 & : ?x \mapsto \text{ex : a} \\
\mu_2 & : ?y \mapsto \text{ex : b} \quad \checkmark
\end{align*}
\]
Union of Solutions (2)

Union of two compatible solutions $\mu_1$ and $\mu_2$:

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

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

- Next lecture: Evaluation of the main algebra operators
Agenda

1. Recap
2. Output Formats
3. SPARQL Semantics
4. Transformation of Queries into Algebra Objects
5. Evaluation of the SPARQL Algebra
6. Summary
Summary

- SPARQL queries are translated into algebra objects
- The BGPs generate solutions
- Other algebra operators combine solutions
- Details in the next lecture