TECHNISCHE UNIVERSITÄT DRESDEN

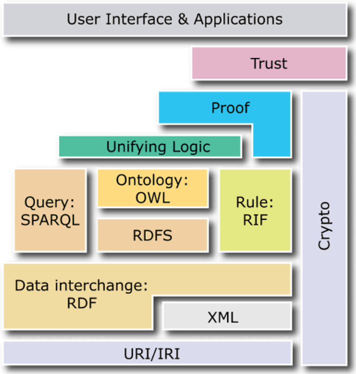
FOUNDATIONS OF SEMANTIC WEB TECHNOLOGIES

Semantics of SPARQL

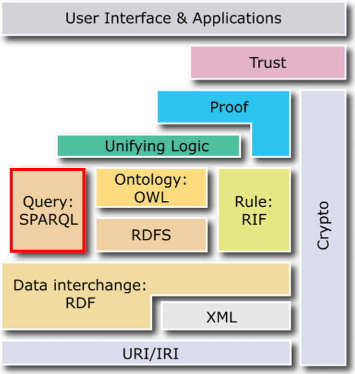
Sebastian Rudolph

DRESDEN concept
Exzellenz aus Wissenschaft und Kultur

The SPARQL Query Language



The SPARQL Query Language



Agenda

Agenda

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

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

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

Output Format CONSTRUCT

CONSTRUCT creates an RDF graph for the results

Example Query

```
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:firstname ?gname .
  ?x foaf:surname   ?fname  }
```

CONSTRUCT Templates with Blank Nodes

Resulting RDF graph

```
@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" .
```

TECHNISCHE UNIVERSITÄT DRESDEN

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

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?”

$\rightsquigarrow$ 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

$\rightsquigarrow$ OWL DL and RDF(S) as query languages

$\rightsquigarrow$ conjunctive queries

Semantics of Query Languages (2)

Query Algebra

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

$\rightsquigarrow$ Relational algebra for SQL

$\rightsquigarrow$ SPARQL Algebra

Agenda

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)
}
```

TECHNISCHE UNIVERSITÄT DRESDEN

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

Semantics of the SPARQL Algebra Operations

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

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)

Semantics of the SPARQL Algebra Operations

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

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)

- Only Bgp(·) matches or evaluates graph patterns

$\leadsto$ 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

$\rightsquigarrow$ How can we formally describe the “results”?

SPARQL Results

Intuition: Results coded as tables of variable assignments

Result:

List of solutions (solution sequence)

$\rightsquigarrow$ 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 ∪ blank nodes ∪ 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 μ is a solution for Bgp(P) over the queried (active) graph G if:

1. the domain of μ is exactly the set of variables in P,
2. there exists an assignment σ 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 $[\![\text{Bgp}(P)]\!]_\text{G}$
- The result is a multi set of solutions μ
- The multiplicity of each solution μ corresponds to the number of different assignments σ

Multi Sets

Definition (Multi Set)

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

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

- We represent a multi se 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: $\dot{\{}a, b, b\dot{\}}$ 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

```
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`)

μ_1: ?who $\mapsto$ ex : Birte, ?what $\mapsto$ "SPARQL"

σ_1: _:x $\mapsto$ _:a

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`)

μ_1: ?who ↦ ex : Birte, ?what ↦ "SPARQL"

σ_1: _:x ↦ _:a

μ_2: ?who ↦ ex : Sebastian, ?what ↦ "DLs and OWL"

σ_2: _:x ↦ _: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`)

μ_1 : ?who ↦ ex : Birte, ?what ↦ "SPARQL"

σ_1 : _:x ↦ _:a

μ_2 : ?who ↦ ex : Sebastian, ?what ↦ "DLs and OWL"

σ_2 : _:x ↦ _:b

Two solutions each with multiplicity 1

Exercise Solution Sets

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

Bgp(`?who ex:gives _:x .  _:x 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" .
```

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

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`)

μ_1 : `?who` ↦ `ex:Birte`,

σ_1 : `_:x` ↦ `_:a` `_:y` ↦ `"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`)

μ_1: ?who ↦ ex : Birte,

σ_1: _:x ↦ _:a   _:y ↦ "SPARQL"

μ_2: ?who ↦ ex : Birte,

σ_2: _:x ↦ _:b   _:y ↦ "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" .
```

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

μ_1 : ?who ↦ ex : Birte,

σ_1 : _:x ↦ _:a _:y ↦ "SPARQL"

μ_2 : ?who ↦ ex : Birte,

σ_2 : _:x ↦ _:b _:y ↦ "SPARQL Algebra"

One solution with multiplicity 2

Union of Solutions (1)

Definition (Compatibility)

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

Union of Solutions (1)

Definition (Compatibility)

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

μ_1: $\texttt{?x} \mapsto \texttt{ex:a}, \texttt{?y} \mapsto \texttt{ex:b}$
μ_2: $\texttt{?y} \mapsto \texttt{ex:b}, \texttt{?z} \mapsto \texttt{ex:c}$

Union of Solutions (1)

Definition (Compatibility)

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

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

Union of Solutions (1)

Definition (Compatibility)

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

μ_1: $?\mathtt{x} \mapsto \mathtt{ex:a}, ?\mathtt{y} \mapsto \mathtt{ex:b}$
μ_2: $?\mathtt{y} \mapsto \mathtt{ex:b}, ?\mathtt{z} \mapsto \mathtt{ex:c}$ ✓

μ_1: $?\mathtt{x} \mapsto \mathtt{ex:a}, ?\mathtt{y} \mapsto \mathtt{ex:b}$
μ_2: $?\mathtt{x} \mapsto \mathtt{ex:b}, ?\mathtt{z} \mapsto \mathtt{ex:c}$

Union of Solutions (1)

Definition (Compatibility)

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

μ_1: $\texttt{?x} \mapsto \texttt{ex:a}, \texttt{?y} \mapsto \texttt{ex:b}$
μ_2: $\texttt{?y} \mapsto \texttt{ex:b}, \texttt{?z} \mapsto \texttt{ex:c}$ ✓

μ_1: $\texttt{?x} \mapsto \texttt{ex:a}, \texttt{?y} \mapsto \texttt{ex:b}$
μ_2: $\texttt{?x} \mapsto \texttt{ex:b}, \texttt{?z} \mapsto \texttt{ex:c}$ ↯

Union of Solutions (1)

Definition (Compatibility)

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

μ_1: $\texttt{?x} \mapsto \texttt{ex:a}, \texttt{?y} \mapsto \texttt{ex:b}$
μ_2: $\texttt{?y} \mapsto \texttt{ex:b}, \texttt{?z} \mapsto \texttt{ex:c}$ ✓

μ_1: $\texttt{?x} \mapsto \texttt{ex:a}, \texttt{?y} \mapsto \texttt{ex:b}$
μ_2: $\texttt{?x} \mapsto \texttt{ex:b}, \texttt{?z} \mapsto \texttt{ex:c}$ ↯

μ_1: $\texttt{?x} \mapsto \texttt{ex:a}$
μ_2: $\texttt{?y} \mapsto \texttt{ex:b}$

Union of Solutions (1)

Definition (Compatibility)

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

μ_1: $?\mathtt{x} \mapsto \mathtt{ex:a}, ?\mathtt{y} \mapsto \mathtt{ex:b}$
μ_2: $?\mathtt{y} \mapsto \mathtt{ex:b}, ?\mathtt{z} \mapsto \mathtt{ex:c}$ ✓

μ_1: $?\mathtt{x} \mapsto \mathtt{ex:a}, ?\mathtt{y} \mapsto \mathtt{ex:b}$
μ_2: $?\mathtt{x} \mapsto \mathtt{ex:b}, ?\mathtt{z} \mapsto \mathtt{ex:c}$ ↯

μ_1: $?\mathtt{x} \mapsto \mathtt{ex:a}$
μ_2: $?\mathtt{y} \mapsto \mathtt{ex:b}$ ✓

Union of Solutions (2)

Union of two compatible solutions μ_1 and μ_2:

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

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

- Next lecture: Evaluation of the main algebra operators

Agenda

Summary

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