1 Introduction

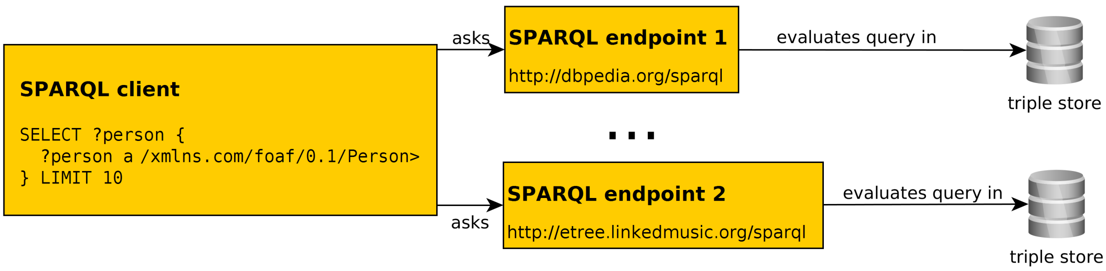
1.1 SPARQL

A simple SPARQL Query

```
SELECT ?person {
  ?person a <http://xmlns.com/foaf/0.1/Person> .
} LIMIT 10
```

To be queried over RDF data inside a **SPARQL endpoint**, e.g. `http://dbpedia.org/sparql`

SPARQL idea



SPARQL Factsheet

- SPARQL 1.1 – 12 W3C Recommendations on 21 March 2013, covering
  - a query language (SPARQL 1.1 Query Language) [**Harris:13:SQL**]
  - an update language (SPARQL 1.1. Update language)
  - SPARQL services (protocol over HTTP, graph management HTTP protocol),
  - an extension for executing distributed queries over more SPARQL endpoints [**Aranda:13:SFQ**]
  - JSON, CSV, TSV, XML query result formats [**Seaborne:13:SQR**]
  - definition of entailment regimes for RDF extensions (e.g. OWL, more in lecture 10) [**Ogbuji:13:SER**].

SPARQL for RDF is like SQL for RDBMS

'Get projects having male administrators starting on the letter N'

```
SELECT e.surname AS es,
       p.name AS pn
FROM employee e, project p
WHERE e.gender = 'male'
      AND p.administratorId = e.id
      AND e.surname LIKE 'N\%';
```

```
PREFIX : <http://example.org/>
SELECT ?sn, (?projname AS ?pn)
WHERE {
  ?e a :Employee .
  ?e :surname ?sn .
  ?e :gender 'male'.
  ?p a :Project .
  ?p :name ?pn .
  ?p :administrator ? e.
  FILTER (strstarts(?sn,'N'))
}
```

However, SPARQL is less powerful comparing to SQL in terms of built-in functions

Is SPARQL the only one ?

Some previous attempts to query SPARQL include:

reactive-rule languages – e.g. Algea

path-based languages – e.g. Versa

relational-based – TRIPLE, Xcerpt, SeRQL

At present

SPARQL is **The standard** for querying RDF. In addition, graph languages have been gaining popularity for querying RDF (e.g. GraphQL `https://graphql.org/`).

1.1.1 SPARQL Basics

Query Types

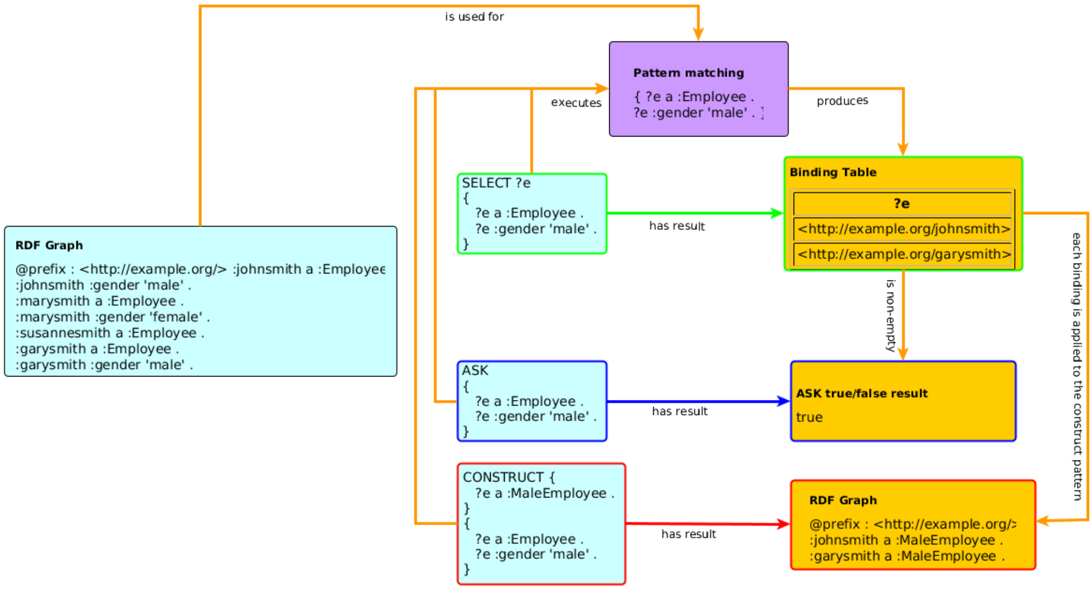
SELECT – returns a binding table (similarly to SQL)

ASK – returns a true/false indicating existence of the given pattern in the RDF graph

CONSTRUCT – returns an RDF graph constructed from the binding table

DESCRIBE – returns an RDF graph describing the given resource (semantics not fixed)

Query Evaluation



Basic Definitions (1)

RDF Term $\in T = T_I \cup T_B \cup T_L$, being a union of set of all IRIs, blank nodes and literals respectively.

example

$\_{:}a \qquad <\texttt{http://example.org/data/John}> \qquad "John"@en$

solution is a mapping $\mu : V \to T$ assigning an RDF term to each variable from the query,

example

$$\mu = \{(?person \to <\texttt{http://example.org/data/John}>), \\ (?personName \to "John"@en)\}$$

result set is a list $R = (\mu_1, \ldots, \mu_n)$ of solutions,

Basic Definitions (2)

triple pattern (TP) is a member of $(T \cup V) \times (T_I \cup V) \times (T \cup V)$,

example

$$(?person, a, \texttt{foaf:Person})$$

or in the turtle syntax

$$?person \quad a \quad \texttt{foaf:Person}$$

basic graph pattern (BGP) is a set $BGP = \{TP_1, \ldots, TP_n\}$ of triple patterns.

example

$?person \quad a \quad \texttt{foaf:Person}. \quad ?person \quad \texttt{rdfs:label} \quad ?label.$

graph store is a mutable container providing an RDF dataset at each time,

Basic Graph Patterns

Repository content:

```
@prefix : <http://example.org/>
@prefix r: <http://dbpedia.org/resource/>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>
:inventors {
  r:Thomas_Edison :invented :bulb .
  r:J_Cimrman :invented :bulb .
  :bulb rdfs:label "Bulb"@en , "Zarovka"@cs .
  :wheel rdfs:label "Wheel"@en .
  _:x :invented :wheel .
  _:y :invented :SteamEngine .
  _:z :invented :Gunpowder .
  :Gunpowder rdfs:label "Strelny prach"@cs .
}
```

Query with a BGP

```
PREFIX : <http://example.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?s ?l
WHERE {
  ?s :invented ?i.
  ?i rdfs:label ?l.}
```

Table 1.1: Result set

s	l
r:Thomas_Edison	“Bulb”@en
r:J_Cimrman	“Bulb”@en
r:Thomas_Edison	“Zarovka”@cs
r:J_Cimrman	“Zarovka”@cs
_:a	“Wheel”@en
_:b	“Strelny prach”@cs

Filtering results

Description

syntax BGP1 **FILTER**(boolean condition) BGP1

description **FILTER** clause filters BGP results (anywhere in a BGP)

Query with a BGP

```
PREFIX : <http://example.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?s ?l
WHERE {
  ?s :invented ?i.
  ?i rdfs:label ?l
  FILTER(regex(?l,"^.ul.*")
    && contains(str(?s),"Cimr"))
}
```

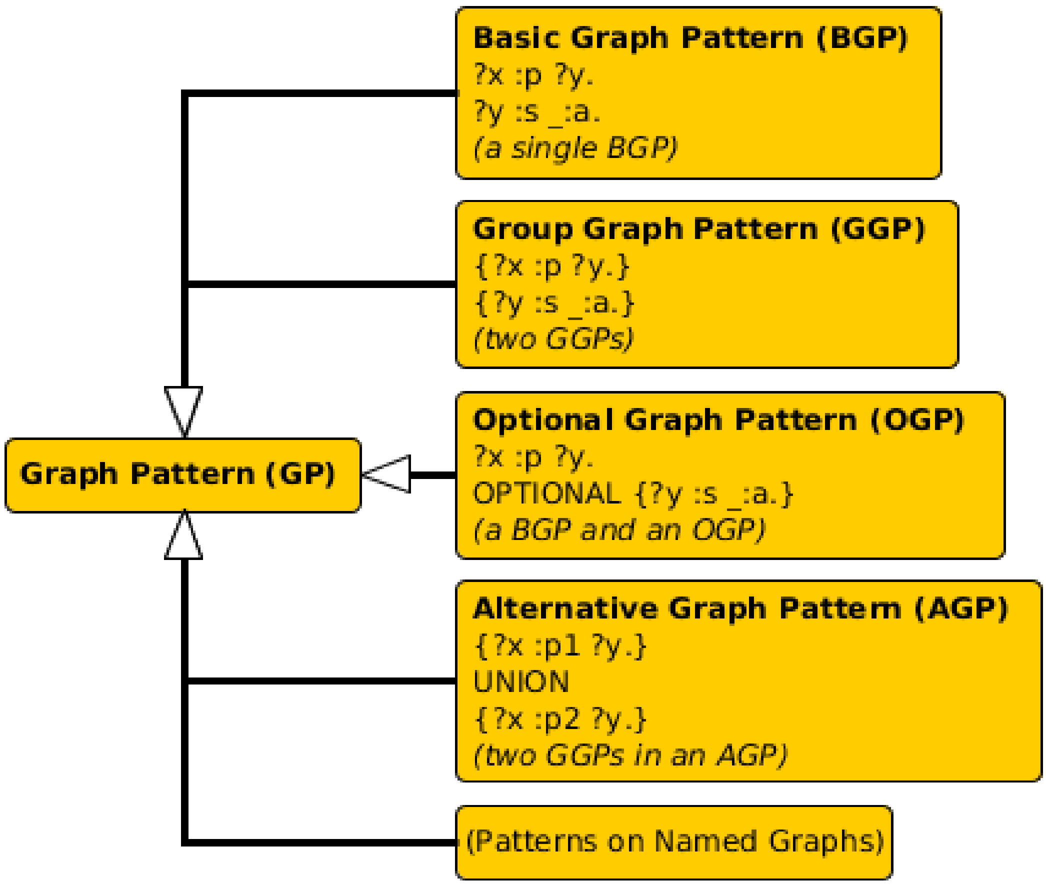
See SPARQL 1.1 spec. `https://www.w3.org/TR/2013/REC-sparql11-query-20130321/#func-rdfTerms`

- string functions – e.g. `strlen, contains, substr, concat, regex, replace`
- RDF term functions – e.g. `isIRI, IRI, isBlank, BNODE, isLiteral, str, lang, datatype`

Graph Patterns – Overview

Graph patterns cover all basic algebraic operations:

- conjunction (sequence of graph patterns),
- disjunction (**`UNION`** pattern),
- negation (**`FILTER NOT EXISTS`**, **`MINUS`**)
- conditional conjunction (**`OPTIONAL`**)



Optional data

Description

syntax GP1 **OPTIONAL** { GP2 }

description results of GP1 are optionally augmented with results of GP2, if any. Optionals are left-associative.

Two optionals

```
PREFIX : <http://example.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?s ?i ?l
WHERE {
    ?s :invented ?i.
    OPTIONAL {
      ?i rdfs:label ?l FILTER (lang(?l)="en").
    } OPTIONAL {
```

```
        ?i rdfs:label ?l FILTER (lang(?l)="cs")
    }
}
```

Table 1.2: Result set

s	l
r:Thomas_Edison	“Bulb”@en
r:J_Cimrman	“Bulb”@en
_:a	“Wheel”@en
_:b	
_:c	“Strelny prach”@cs

Other examples

FILTERing with regular expressions

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
SELECT ?title
WHERE { ?x dc:title ?title .
?x dc:author ?author
FILTER regex(?title, ".SPARQL") }
```

Order of OPTIONALs might be important

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX books: <http://books.example.org>
SELECT ?writing ?name
WHERE
{ ?writing rdf:type books:Essay .
OPTIONAL { ?writing books:translator ?p . ?p dc:name ?name . } .
OPTIONAL { ?writing books:author ?p . ?p dc:name ?name . }}
```

Negation

negation as failure – i.e. what cannot be inferred is considered false.

two constructs – **MINUS** vs. **FILTER NOT EXISTS**

MINUS

```
PREFIX : <http://example.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?s1 ?i
{   ?s1 :invented ?i.
    MINUS {
      ?s2 :invented ?i .
      FILTER(?s1 != ?s2) . }}
```

Variable `?s1` is not bound in the **MINUS** pattern. Returns all inventors. FILTER NOT EXISTS

```
PREFIX : <http://example.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?s1 ?i
{
    ?s1 :invented ?i.
    FILTER NOT EXISTS {
      ?s2 :invented ?i .
      FILTER(?s1 != ?s2). }}
```

Returns all inventions that were invented just by one inventor.

Property Paths

Description

Property paths allow to express simple regular expressions on properties, as follows

syntax	matches ($e_{(i)}$ means path element, $p_{(i)}$ means iri or ˆiri)
iri	an IRI (path of length 1)
ˆe	an inverse path (o → s)
e_1 / e_2	a sequence path of e_1 followed by e_2
e_1 | e_2	an alternative path of e_1 or e_2
$e*$	a sequence path of zero or more matches of e
$e+$	a sequence path of one or more matches of e
$e?$	a sequence path of zero or one more matches of e
$!(p_1\|\ldots\|p_n)$	any IRI not matching any of p_i
(e)	group path (brackets for precedence)

Property Paths – Examples

Get the name of a resource

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT *
{
    ?s rdfs:label|dc:title ?name.
}
```

Get elements of an RDF collection

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT *
{
    ?s (rdf:rest*)/rdf:first ?listItem.
}
```

Aggregations

Description

Similarly to SQL, SPARQL allows using aggregation functions for numeric/sting data:

COUNT(?var), or COUNT(DISTINCT ?var) – counts number of (distinct) occurences of `?var` in the resultset,

MIN(?v),MAX(?v),SUM(?v),AVG(?v) – analogous to their SQL counterparts,

GROUP_CONCAT(?var; separator = <SEP>) AS ?group) – concatenates all elements in the group with the given separator character,

SAMPLE – takes an arbitrary representative from the group.

Usage of `(?expr as ?var)` alias is obligatory.

Similarly to SQL, SPARQL allows computing aggregates over particular data groups and filter in them using **GROUP BY**/**HAVING** construct.

Aggregation – Examples

Compute the number of inventors of each invention.

```
PREFIX : <http://example.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT (COUNT(?s) as ?count) ?i (GROUP_CONCAT(?s;separator=",") as ?inventors)
FROM :inventors
WHERE {
    ?s :invented ?i.
}
GROUP BY ?i
HAVING (COUNT(?s) > 1)
```

Variable assignment

Description

Variables can be assigned results of function (or aggregation function). The syntax is `(expr AS ?v)`, where `expr` is an expression and `?v` is the newly create variable not appearing before.

Compute the number of inventions of each inventor.

```
PREFIX : <http://example.org/>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT (COUNT(?s) AS ?count) ?invention
FROM :inventors
WHERE {
    ?s :invented ?i .
    ?i rdfs:label ?l
    BIND (concat("Invention: ",?l) AS ?invention)
}
GROUP BY ?i ?invention
```

Distributed Queries

Syntax and semantics

syntax ...**SERVICE** (**SILENT**) sparqlServiceURI { GP }

semantics this clause poses a sparql query described by graph pattern GP to a remote SPARQL endpoint *sparqlServiceURI*

DBPedia service query

```
PREFIX : <http://example.org/>
PREFIX p: <http://dbpedia.org/property/>
PREFIX r: <http://dbpedia.org/resource/>
SELECT ?s ?p ?o ?i
WHERE {
  GRAPH :inventors { ?s :invented ?i. }
  OPTIONAL { SERVICE SILENT
       <http://dbpedia.org/sparql> {
     ?s ?p ?o
      FILTER( strstarts(str(?p),
            concat(str(p:),"death")) ) }}}
```

Local repo content

```
@prefix : <http://example.org/>
@prefix p: <http://dbpedia.org/property/>
@prefix r: <http://dbpedia.org/resource/>
:inventors {
  r:Thomas_Edison :invented :bulb.
  r:J_Cimrman :invented :bulb.
}
```

Selected Other Features

- **VALUES** – predefined variable bindining specified in the tabular form
- **ORDER BY, LIMIT, OFFSET** – used analogously to SQL
- **FROM, FROM NAMED** – used to specify active default/named graphs for the query
- **SELECT DISTINCT** – removes duplicates from the results

SPARQL Entailment Regimes

- simply – SPARQL spec. [**Harris:13:SQL**] defines evaluation of BGPs w.r.t. *simple entailment*
- [**Ogbuji:13:SER**] defines a several other entailment regimes for SPARQL BGPs:

  RDF entailment, RDFS entailment, D-entailment , as defined in RDF spec.

  OWL 2 entailments, RIF entailment , that are the much more expressive, see lecture 10.

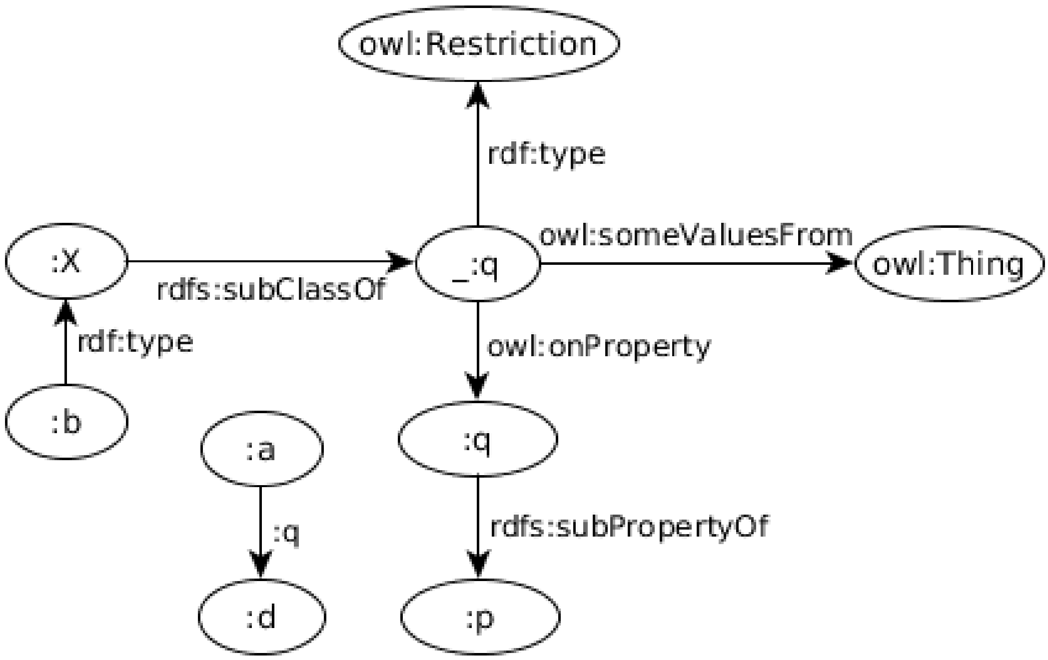
  ... conditions for defining custom entailment regimes

All SPARQL entailment regimes must ensure

- compliance with the corresponding entailment (e.g. RDF, RDFS)
- finiteness of results

- only *canonical* b-nodes can be returned (ensured by skolemization of both the query and the queried graph),
- only finite part of respective vocabularies can be returned as query results (e.g. RDF vocabulary without `rdf:_n` properties not occuring in the graph).

SPARQL Evaluation Semantics




```
PREFIX : <http://ex.org/e1>
SELECT ?x
WHERE { ?x :p :d }
```

Simple-entailment No result.

RDF-entailment No result.

RDFS-entailment One result: `?x=:a`.

OWL-entailment Two results: `?x=:a` and `?x=:b`.

SPARQL SELECT/ASK results

CSV for **`SELECT`**; loses information about datatypes/languages of RDF terms

TSV for **`SELECT`**; is lossless

XML, JSON for **SELECT,ASK**; is lossless, supports additional information (e.g. columns identification through *link* attribute),

```
{
  "head": {
    "vars": [ "person", "name" ]
  },
  "results": {
    "bindings":
     [{
      "person": {
         "type": "uri",
         "value": "http://ex.com/p1" },
      "name": {
         "type":"literal",
         "value": "Smith" }
     },{
      "person": {
         "type": "uri",
         "value": "http://ex.com/p2" }
     }]
  },
}
```

Related Technologies

SPIN (SPARQL inference notation) – SPARQL rules encoded in RDF (`http://spinrdf.org/`)

iSPARQL – SPARQL visual query builder (`http://oat.openlinksw.com/isparql/`)

SNORQL – Web front-end for exploring SPARQL endpoints (`https://github.com/kurtjx/SNORQL`)

SeRQL – Sesame query language (alternative to SPARQL)

SQWRL (Semantic Query-Enhanced Web Rule Language) – query language based on SWRL (see next lecture), `http://protege.cim3.net/cgi-bin/wiki.pl?SQWRL`

1.1.2 SPARQL Update (Graph Update Operations)

Inserting

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT { <http://example/person> dc:title  "John" }
WHERE {}
```

Deleting

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
DELETE WHERE
  ?person a foaf:Person .
}
```

Replacing

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX dbo: <http://dbpedia.org/ontology/>
DELETE { ?person a foaf:Person . }
INSERT { ?person a dbo:Person . }
WHERE { ?person a foaf:Person . }
```

Other operations

- LOAD – loading a graph into a graph store
- CLEAR – clearing a graph inside a graph store
- CREATE – create a new graph in a graph store
- DROP – deletes a graph in a graph store
- COPY – inserts all triples from one graph to another, clearing the dest.
- MOVE – moves all triples from one graph to another
- ADD – inserts all triples from one graph to another, keeping the dest.

See `https://www.w3.org/TR/sparql11-update/` for details