SPARQL 1.1, Rules and Graph Database solutions

Ernesto Jiménez-Ruiz, Lecturer in Artificial Intelligence

Before we start...

Laboratory Session

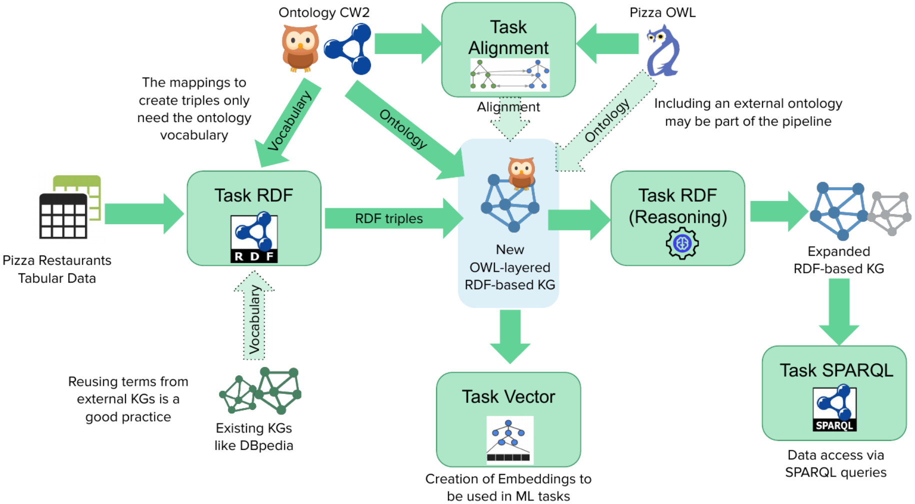
- R201 (Franklin building).
- **Session 2, Weeks 8** (this week) **and 10** officially in R201.

Where are we? Module organization.

✓ Introduction: Becoming a knowledge scientist.
✓ RDF-based knowledge graphs.
✓ OWL ontology language. Focus on modelling.
✓ SPARQL 1.0 Query Language.
✓ From tabular data to KG.
✓ RDFS Semantics and OWL 2 profiles.

6. **SPARQL 1.1, Rules and Graph Database solutions.**(Today)
7. Ontology Alignment.
8. Ontology (KG) Embeddings and Machine Learning.
9. (Large) Language Models and KGs. (Seminar)

The global picture



Coursework part 2

- Sunday, 12 May 2024, 5:00 PM

- Team registration March 17

- Components:
  - ✓ Tabular Data to Knowledge Graph: 40% (Weeks 2 and 5)
  - ✓ SPARQL and Reasoning: 20% (Weeks 4, 7 and 8)
  - Ontology Alignment: 10% (Week 9)
  - Ontology Embeddings: 10% (Week 10)

Learning outcomes for today

– SPARQL 1.1 for the coursework!

– Knowledge of the infrastructure for real-world solutions.

– A bit of Rules and SHACL.

SPARQL 1.1

SPARQL

- SPARQL Protocol And RDF Query Language
- Standard language to query graph data represented as **RDF triples**
- W3C Recommendations
  - **SPARQL 1.0:** W3C Recommendation 15 January 2008
  - **SPARQL 1.1:** W3C Recommendation 21 March 2013

SPARQL

- SPARQL Protocol And RDF Query Language
- Standard language to query graph data represented as **RDF triples**
- W3C Recommendations
  - **SPARQL 1.0:** W3C Recommendation 15 January 2008
  - **SPARQL 1.1:** W3C Recommendation 21 March 2013
- In this lecture we will learn about the extensions in SPARQL 1.1.
- Documentation:
  - Syntax and semantics of the SPARQL query language for RDF.
    `https://www.w3.org/TR/sparql11-overview/`

Recap: Components of a SPARQL (1.0) query

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dbo:  <http://dbpedia.org/ontology/>
SELECT DISTINCT ?costar
WHERE {
    ?jd foaf:name "Johnny Depp"@en .
    ?m dbo:starring ?jd .
    ?m dbo:starring ?other .
    ?other foaf:name ?costar .
    FILTER (STR(?costar)!="Johnny Depp")
}
ORDER BY ?costar
```

Recap: Components of a SPARQL (1.0) query

Prologue: prefix definitions

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dbo:  <http://dbpedia.org/ontology/>
SELECT DISTINCT ?costar
WHERE {
    ?jd foaf:name "Johnny Depp"@en .
    ?m dbo:starring ?jd .
    ?m dbo:starring ?other .
    ?other foaf:name ?costar .
    FILTER (STR(?costar)!="Johnny Depp")
}
ORDER BY ?costar
```

Recap: Components of a SPARQL (1.0) query

Results: (1) variable list, (2) query type (SELECT, ASK, CONSTRUCT, DESCRIBE), (3) remove duplicates (DISTINCT, REDUCED)

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dbo:  <http://dbpedia.org/ontology/>
SELECT DISTINCT ?costar
WHERE {
    ?jd foaf:name "Johnny Depp"@en .
    ?m dbo:starring ?jd .
    ?m dbo:starring ?other .
    ?other foaf:name ?costar .
    FILTER (STR(?costar)!="Johnny Depp")
}
ORDER BY ?costar
```

Recap: Components of a SPARQL (1.0) query

Query pattern: graph pattern to be matched + filters

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dbo:  <http://dbpedia.org/ontology/>
SELECT DISTINCT ?costar
WHERE {
    ?jd foaf:name "Johnny Depp"@en .
    ?m dbo:starring ?jd .
    ?m dbo:starring ?other .
    ?other foaf:name ?costar .
    FILTER (STR(?costar)!="Johnny Depp")
}
ORDER BY ?costar
```

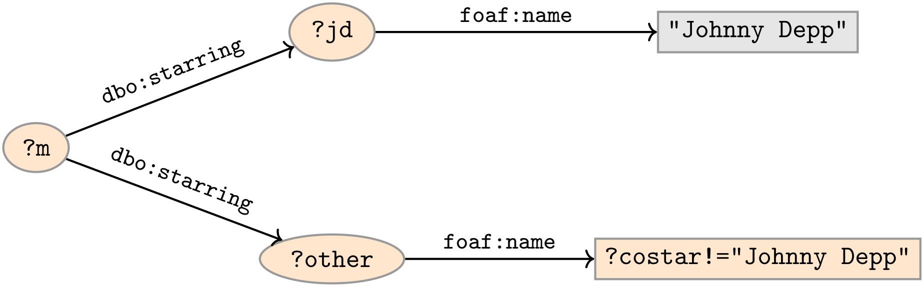
Recap: Components of a SPARQL (1.0) query

Solution modifiers: ORDER BY, LIMIT, OFFSET

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dbo:  <http://dbpedia.org/ontology/>
SELECT DISTINCT ?costar
WHERE {
    ?jd foaf:name "Johnny Depp"@en .
    ?m dbo:starring ?jd .
    ?m dbo:starring ?other .
    ?other foaf:name ?costar .
    FILTER (STR(?costar)!="Johnny Depp")
}
ORDER BY ?costar
```

Recap: Graph Patterns

The previous SPARQL query pattern as a graph:



Pattern matching: assign values to variables (or blank nodes) to make this a sub-graph of the RDF graph.

SPARQL 1.1: new features

- The new features in **SPARQL 1.1 QUERY language**:
  - Assignments and Expressions
  - Aggregates
  - Subqueries
  - Negation (new syntax)
  - Property paths

SPARQL 1.1: new features

- The new features in **SPARQL 1.1 QUERY language**:
  - Assignments and Expressions
  - Aggregates
  - Subqueries
  - Negation (new syntax)
  - Property paths
- Specification for:
  - **SPARQL 1.1 UPDATE Language**
  - **SPARQL 1.1 Federated Queries**
  - **SPARQL 1.1 Entailment Regimes**

Assignment and Expressions

– The value of an expression can be assigned/bound to a new variable
– Can be used in SELECT, BIND or GROUP BY clauses:
  (expression AS ?var)

Expressions in SELECT clause

```
SELECT ?city (xsd:integer(?pop)/xsd:float(?area) AS ?density)
{
  ?city dbo:populationTotal ?pop .
  ?city dbo:PopulatedPlace/areaTotal ?area .
  ?city dbo:country dbr:United_Kingdom .
  FILTER (xsd:float(?area)>0.0)
}
```

Aggregates: Grouping and Filtering

- Solutions can optionally be grouped according to one or more expressions.
- To specify the group, use `GROUP BY`.
- If `GROUP BY` is not used, then **only one (implicit) group**

Aggregates: Grouping and Filtering

- Solutions can optionally be grouped according to one or more expressions.
- To specify the group, use `GROUP BY`.
- If `GROUP BY` is not used, then **only one (implicit) group**
- To filter solutions resulting from grouping, use `HAVING`.
- `HAVING` operates over grouped solution sets, in the same way that FILTER operates over un-grouped ones.

Aggregates: Example

Actors with more than 15 movies

```
SELECT ?name (COUNT(?movie) AS ?mcount)
WHERE {
    ?actor foaf:name ?name .
    ?movie dbo:starring ?actor .
}
GROUP BY ?name
HAVING (COUNT(?movie) > 15)
ORDER BY DESC (?mcount)
```

Aggregates: Example

Actors with more than 15 movies

```
SELECT ?name (COUNT(?movie) AS ?mcount)
WHERE {
    ?actor foaf:name ?name .
    ?movie dbo:starring ?actor .
}
GROUP BY ?name
HAVING (COUNT(?movie) > 15)
ORDER BY DESC (?mcount)
```

† Only expressions consisting of aggregates and constants may be projected, together with variables in `GROUP BY`.

Aggregates: common functions

- `Count` counts the number of times a variable has been bound.
- `Sum` sums numerical values of bound variables.
- `Avg` finds the average of numerical values of bound variables.
- `Min` finds the minimum of the numerical values of bound variables.
- `Max` finds the maximum of the numerical values of bound variables.

† Aggregates assume CWA and UNA

Subqueries

Subqueries

- A way to embed SPARQL queries within other queries
- Subqueries are evaluated first and the results are projected to the outer query.

Subqueries

- A way to embed SPARQL queries within other queries
- Subqueries are evaluated first and the results are projected to the outer query.

```
SELECT ?country ?pop (round(?pop/?worldpop*1000)/10 AS ?percentage) WHERE {
  ?country rdf:type dbo:Country .
  ?country dbo:populationTotal ?pop .
  {
    SELECT (sum(?p) AS ?worldpop) WHERE {
      ?c rdf:type dbo:Country .
      ?c dbo:populationTotal ?p .}
  }
}  ORDER BY desc(?pop)
```

Subqueries

- A way to embed SPARQL queries within other queries
- Subqueries are evaluated first and the results are projected to the outer query.

```
SELECT ?country ?pop (round(?pop/?worldpop*1000)/10 AS ?percentage) WHERE {
  ?country rdf:type dbo:Country .
  ?country dbo:populationTotal ?pop .
  {
    SELECT (sum(?p) AS ?worldpop) WHERE {
      ?c rdf:type dbo:Country .
      ?c dbo:populationTotal ?p .}
  }
}  ORDER BY desc(?pop)
```

† Note that the *Sum()* aggregation is done over all the elements (single default group).

Negation in SPARQL 1.0

COMBINING OPTIONAL, FILTER and !BOUND:

People without names

```
SELECT DISTINCT * WHERE {
    ?person a foaf:Person .
    OPTIONAL {
        ?person foaf:name ?name .
    FILTER (!bound(?name))
    }
}
```

However, this is not very easy to write and interpret.

Negation in SPARQL 1.1: MINUS and FILTER NOT EXISTS

Two ways to do negation. *e.g.*, retrieve people without a name:

```
SELECT DISTINCT * WHERE {
    ?person a foaf:Person .
    MINUS { ?person foaf:name ?name }
}

SELECT DISTINCT * WHERE {
    ?person a foaf:Person .
    FILTER NOT EXISTS { ?person foaf:name ?name }
}
```

†Negation assumes CWA and UNA

Property paths

Property paths: basic motivation

- Some queries get needlessly large.
- SPARQL 1.1 define a small language to defined paths.
- Examples:
  - `city:ernesto foaf:knows+ ?friend` to extract all friends of friends.
  - `foaf:maker|dc:creator` instead of `UNION`.
  - Friend's names, `{ _:me foaf:knows/foaf:name ?friendsname }`.
  - Sum several items:

```
SELECT (sum(?cost) AS ?total) { :order :hasItem/:price ?cost }
```

Property paths: example

```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
PREFIX dbo:  <http://dbpedia.org/ontology/>
SELECT DISTINCT ?costar
WHERE {
    ?m dbo:starring/foaf:name "Johnny Depp"@en .
    ?m dbo:starring/foaf:name ?costar .
    FILTER (STR(?costar)!="Johnny Depp")
}
ORDER BY ?costar
```

†Similar to blank node syntax.

Property paths: syntax

<table>
<tr><th>Syntax Form</th><th>Matches</th></tr>
<tr><td>iri</td><td>An (property) IRI. A path of length one.</td></tr>
<tr><td>^elt</td><td>Inverse path (object to subject).</td></tr>
<tr><td>elt1 / elt2</td><td>A sequence path of elt1 followed by elt2.</td></tr>
<tr><td>elt1 | elt2</td><td>A alternative path of elt1 or elt2 (all possibilities are tried).</td></tr>
<tr><td>elt*</td><td>Seq. of zero or more matches of elt.</td></tr>
<tr><td>elt+</td><td>Seq. of one or more matches of elt.</td></tr>
<tr><td>elt?</td><td>Zero or one matches of elt.</td></tr>
<tr><td>!iri or !(iri$_1$| ...|iri$_n$)</td><td>Negated property set.</td></tr>
<tr><td>!^iri or !(^iri$_i$| ...|^iri$_n$)</td><td>Negation of inverse path.</td></tr>
<tr><td>!(iri$_1$|...|iri$_j$|^iri$_{j+1}$|...|^iri$_n$)</td><td>Negated combination of forward and inverese properties.</td></tr>
<tr><td>(elt)</td><td>A group path elt, brackets control precedence.</td></tr>
</table>

* elt is a path element, which may itself be composed of path constructs (see Syntax form).

SPARQL 1.1 Entailment Regimes

OWL 2 Entailment regimes: overview

- Gives guidance for SPARQL query engines and expectations to users of SPARQL Endpoints.
- Basic graph pattern by means of subgraph matching: **simple entailment**
- Solutions that **implicitly** follow from the queried graph: **entailment regimes**
- **RDF entailment**, **RDF Schema (RDFS) entailment**, D-Entailment, **OWL 2 RDF-Based Semantics entailment**, **OWL 2 Direct Semantics entailment**, and RIF-Simple entailment
- `https://www.w3.org/TR/2013/REC-sparql11-entailment-20130321/`

Entailment regimes: example (i)

- `ex:book1 rdf:type ex:Publication .`
- `ex:book2 rdf:type ex:Article .`
- `ex:Article rdfs:subClassOf ex:Publication .`
- `ex:publishes rdfs:range ex:Publication .`
- `ex:MITPress ex:publishes ex:book3 .`

Entailment regimes: example (i)

- `ex:book1 rdf:type ex:Publication .`
- `ex:book2 rdf:type ex:Article .`
- `ex:Article rdfs:subClassOf ex:Publication .`
- `ex:publishes rdfs:range ex:Publication .`
- `ex:MITPress ex:publishes ex:book3 .`

QUERY 1: `SELECT ?prop WHERE { ?prop rdf:type rdf:Property . }`

Entailment regimes: example (i)

- `ex:book1 rdf:type ex:Publication .`
- `ex:book2 rdf:type ex:Article .`
- `ex:Article rdfs:subClassOf ex:Publication .`
- `ex:publishes rdfs:range ex:Publication .`
- `ex:MITPress ex:publishes ex:book3 .`

QUERY 1: `SELECT ?prop WHERE { ?prop rdf:type rdf:Property . }`
Results = {} using Simple entailment. Non empty with the other entailments.

Entailment regimes: example (i)

- `ex:book1 rdf:type ex:Publication .`
- `ex:book2 rdf:type ex:Article .`
- `ex:Article rdfs:subClassOf ex:Publication .`
- `ex:publishes rdfs:range ex:Publication .`
- `ex:MITPress ex:publishes ex:book3 .`

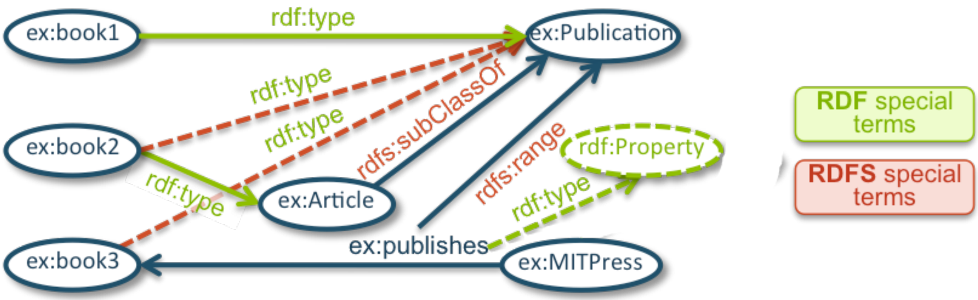
QUERY 2: `SELECT ?pub WHERE { ?pub rdf:type ex:Publication . }`

Entailment regimes: example (i)

- `ex:book1 rdf:type ex:Publication .`
- `ex:book2 rdf:type ex:Article .`
- `ex:Article rdfs:subClassOf ex:Publication .`
- `ex:publishes rdfs:range ex:Publication .`
- `ex:MITPress ex:publishes ex:book3 .`

QUERY 2: `SELECT ?pub WHERE { ?pub rdf:type ex:Publication . }`
Results = {ex:book1} using Simple and RDF entailment.
Results = {ex:book1, book2, book3} with RDFS and OWL entailments.

Entailment regimes: example (ii)



(*) Expansion of the RDF graph. Green dashed lines inferred via **RDF-Semantics** (See rules: `https://www.w3.org/TR/rdf-mt/#RDFRules`). Red-dashed lines via **RDFS-Semantics** (as in Week 7 rules: `https://www.w3.org/TR/rdf-mt/#RDFSRules`)

OWL 2 Entailment regimes: OWL 2 Direct Semantics (i)

– BGP matching has to be defined using semantic entailment.

– Not possible to execute queries over an unique “extended” version of the graph.

  – Unlike in RDFS and OWL 2 RL, graph extension (via reasoning) + then query not possible.

– Models of an ontology may not be finite.

OWL 2 Entailment regimes: OWL 2 Direct Semantics (ii)

Challenges:

- Expressive datatype constructs may lead to infinite answers:
  - i.e., required binding to infinitely many integer values
  - Solution: limit to literals explicitly mentioned in graph
- OWL Direct Semantics defined in terms of OWL objects
  - RDF graph and query must first be translated.
  - Requirement: Restriction on RDF graphs and SPARQL queries

OWL 2 Entailment regimes: OWL 2 Direct Semantics (iii)

Variable typing in both the Graph and the SPARQL query:

– In order to have an unambiguous correspondence between BGPs and OWL objects

– e.g., `?x rdf:type TYPE .`

– `TYPE` one of: owl:Class, owl:ObjectProperty, owl:DataProperty, owl:Datatype, or owl:NamedIndividual

OWL 2 Entailment Regimes: OWL 2 RDF-based Semantics

- Direct extension of the RDFS semantics
- It interprets RDF triples directly without the need of mapping an RDF graph into OWL objects
- Treats classes as individuals that refer to elements of the domain
- Reasoning under the OWL 2 RDF-Based Semantics is semidecidable.
- Guaranteed termination might be achieved by returning an incomplete solution for certain queries.

OWL 2 Entailment Regimes: example (i)

– Graph: `ex:a rdf:type ex:C`
– BGP in query:

```
?x rdf:type
  [
  rdf:type owl:Class ;
  owl:unionOf( ex:C ex:D )
  ]
```

OWL 2 Entailment Regimes: example (i)

- Graph: `ex:a rdf:type ex:C`
- BGP in query:

```
?x rdf:type
  [
  rdf:type owl:Class ;
  owl:unionOf( ex:C ex:D )
  ]
```

- `ex:a` not returned in the solution for `?x` using OWL 2 RDF-Based Semantics
  - The Graph does not include that this union is the class extension of any domain element

OWL 2 Entailment Regimes: example (ii)

- Graph: `ex:a rdf:type ex:C`
- BGP in query:

```
?x rdf:type
  [
  rdf:type owl:Class ;
  owl:unionOf( ex:C ex:D )
  ]
```

- `ex:a` would be a solution for `?x` using OWL 2 Direct Semantics
  - classes denote sets and not domain elements

OWL 2 Entailment regimes: overview

OWL 2 RDF-based Semantics Entailment Regime

– Direct extension of the RDFS semantics
– Interprets RDF triples directly without the need of mapping an RDF graph into OWL objects.
– Incomplete for OWL 2 and undecidable for OWL 2 Full.

OWL 2 Entailment regimes: overview

OWL 2 RDF-based Semantics Entailment Regime

- Direct extension of the RDFS semantics
- Interprets RDF triples directly without the need of mapping an RDF graph into OWL objects.
- Incomplete for OWL 2 and undecidable for OWL 2 Full.

OWL 2 Direct Semantics Entailment Regime

- Decidable if some restrictions are imposed to the RDF graph and SPARQL queries.
- Implementation using HermiT:
  - Ilianna Kollia and Birte Glimm. **Using SPARQL with RDFS and OWL entailment**.
  - BGP-OWL: `https://github.com/iliannakollia/owl-bgp`

OWL 2 Entailment Regimes: Complexity and Profiles

– Entailment under **OWL 2 (DL) Direct Semantics** entailment is decidable, but computationally hard.
– Entailment under **OWL 2 (DL) RDF-based semantics** is incomplete for OWL 2 and undecidable for OWL 2 Full.
– No Direct Semantics for OWL 2 Full.

OWL 2 Entailment Regimes: Complexity and Profiles

- Entailment under **OWL 2 (DL) Direct Semantics** entailment is decidable, but computationally hard.
- Entailment under **OWL 2 (DL) RDF-based semantics** is incomplete for OWL 2 and undecidable for OWL 2 Full.
- No Direct Semantics for OWL 2 Full.
- **OWL 2 Profiles**:
  - **Direct Semantics for OWL 2 QL and EL** Profiles have very nice computational properties.
  - Entailment under **OWL 2 QL and EL RDF-based semantics** is incomplete as well.

OWL 2 Entailment Regimes: Complexity and Profiles (cont.)

- **OWL 2 RL** defines a syntactic subset of OWL 2. For RDF graphs that fall into this syntactic subset:
  - ✓ Direct Semantics and RDF-based Semantics yield the same (**complete and sound**) results.

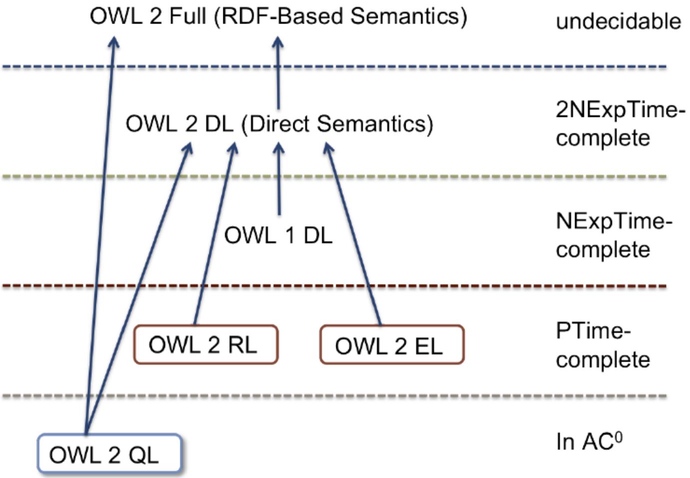
OWL 2 Entailment Regimes: Complexity and Profiles (cont.)

- **OWL 2 RL** defines a syntactic subset of OWL 2. For RDF graphs that fall into this syntactic subset:
  - ✓ Direct Semantics and RDF-based Semantics yield the same (**complete and sound**) results.
  - For Direct Semantics the input RDF graph has to satisfy some constrains.

OWL 2 Entailment Regimes: Complexity and Profiles (cont.)

- **OWL 2 RL** defines a syntactic subset of OWL 2. For RDF graphs that fall into this syntactic subset:
  - ✓ Direct Semantics and RDF-based Semantics yield the same (**complete and sound**) results.
  - For Direct Semantics the input RDF graph has to satisfy some constrains.
  - ✓ The RDF-Based semantics can be use with any RDF graph.

OWL 2 Entailment Regimes: Complexity and Profiles (cont.)



Entailment Regimes: Service description

– How do we know the Entailment Regime for a SPARQL endpoint?

Entailment Regimes: Service description

– How do we know the Entailment Regime for a SPARQL endpoint?
– SPARQL 1.1 Service Descriptions (triples)

Entailment Regimes: Service description

- How do we know the Entailment Regime for a SPARQL endpoint?
- SPARQL 1.1 Service Descriptions (triples)
- Predicates `sd:defaultEntailmentRegime` or `sd:entailmentRegime`:
  - e.g.: `http://dbpedia.org/sparql sd:entailmentRegime er:OWL-Direct .`

Entailment Regimes: Service description

- How do we know the Entailment Regime for a SPARQL endpoint?
- SPARQL 1.1 Service Descriptions (triples)
- Predicates `sd:defaultEntailmentRegime` or `sd:entailmentRegime`:
  - e.g.: `http://dbpedia.org/sparql sd:entailmentRegime er:OWL-Direct .`
- Predicates `sd:defaultSupportedEntailmentProfile` or `sd:supportedEntailmentProfile`:
  - e.g.: `http://dbpedia.org/sparql sd:supportedEntailmentProfile owlp:RL .`

Entailment Regimes: Service description

- How do we know the Entailment Regime for a SPARQL endpoint?
- SPARQL 1.1 Service Descriptions (triples)
- Predicates `sd:defaultEntailmentRegime` or `sd:entailmentRegime`:
  - e.g.: `http://dbpedia.org/sparql sd:entailmentRegime er:OWL-Direct .`
- Predicates `sd:defaultSupportedEntailmentProfile` or `sd:supportedEntailmentProfile`:
  - e.g.: `http://dbpedia.org/sparql sd:supportedEntailmentProfile owlp:RL .`
- Unfortunately this information is not always provided.

Entailment Regimes: Service description (relevant URIs)

- **Simple Entailment:** http://www.w3.org/ns/entailment/Simple
- **RDF Entailment:** http://www.w3.org/ns/entailment/RDF
- **RDFS Entailment:** http://www.w3.org/ns/entailment/RDFS
- **D Entailment:** http://www.w3.org/ns/entailment/D
- **OWL Entailment with Direct Semantics:** http://www.w3.org/ns/entailment/OWL-Direct
- **OWL Entailment with RDF Based Semantics:** http://www.w3.org/ns/entailment/OWL-RDF-Based
- **RIF Entailment:** http://www.w3.org/ns/entailment/RIF

- **OWL 2 Full**: http://www.w3.org/ns/owl-profile/Full
- **OWL 2 DL**: http://www.w3.org/ns/owl-profile/DL
- **OWL 2 EL**: http://www.w3.org/ns/owl-profile/EL
- **OWL 2 QL**: http://www.w3.org/ns/owl-profile/QL
- **OWL 2 RL**: http://www.w3.org/ns/owl-profile/RL

SPARQL 1.1 Federated Query

Federated query support

– The `SERVICE` keyword instructs a federated query processor to invoke a portion of a SPARQL query against a remote SPARQL service/endpoint.

– Like a remote triple pattern matching.

– SPARQL service: any implementation conforming to the *SPARQL 1.1 Protocol for RDF*

– Supported in Jena (Java) and GraphDB, but not supported in rdflib (Python).

Federated query support: example (i)

Combining local graph file with remote SPARQL service

```
SELECT ?name
WHERE {
  <http://example.org/mylocalfoaf/I> foaf:knows ?person .
  SERVICE <http://people.example.org/sparql> {
    ?person foaf:name ?name .   }
}
```

Federated query support: example (ii)

Laureates and spouses according to DBpedia

```
SELECT DISTINCT ?label ?spouse
WHERE {
  ?laur rdf:type np:Laureate .
  ?laur rdfs:label ?label .
  ?laur owl:sameAs ?kglau .
  SERVICE <http://dbpedia.org/sparql> {
    ?dblau owl:sameAs ?kglau .
    ?dblau dbo:spouse ?spouse .
  }
}
```

SPARQL over Named Graphs

Recap: 'Graph' Graph Patterns (RDF datasets)

- SPARQL queries are executed against an **RDF dataset**
- An RDF dataset comprises
  - One **default graph** (unnamed) graph.
  - Zero or more **named graphs** identified by an URI

Recap: ’Graph’ Graph Patterns (RDF datasets)

- SPARQL queries are executed against an **RDF dataset**
- An RDF dataset comprises
  - One **default graph** (unnamed) graph.
  - Zero or more **named graphs** identified by an URI
- FROM and FROM NAMED keywords allows to select an RDF dataset
- Keyword GRAPH makes the named graphs the **active graph** for pattern matching

Recap: ’Graph’ Graph Patterns (RDF datasets)

- SPARQL queries are executed against an **RDF dataset**
- An RDF dataset comprises
  - One **default graph** (unnamed) graph.
  - Zero or more **named graphs** identified by an URI
- FROM and FROM NAMED keywords allows to select an RDF dataset
- Keyword GRAPH makes the named graphs the **active graph** for pattern matching
- Not well supported in Jena and rdflib. Supported in GraphDB.

‘Graph’ Graph Patterns: examples (i)

```
SELECT DISTINCT ?person
WHERE {
  GRAPH city:academic-year-2022 {
    ?person rdf:type foaf:Person .
  }
}
```

‘Graph’ Graph Patterns: examples (ii)

```
SELECT DISTINCT ?person
WHERE {
  GRAPH ?named_graph {
    ?person rdf:type foaf:Person .
  }
}
```

‘Graph’ Graph Patterns: examples (iii)

```
SELECT DISTINCT ?person
FROM city:academic-year-2021
FROM city:academic-year-2022
{
  ?person rdf:type foaf:Person .
}
```

SPARQL 1.1 UPDATE Language

SPARQL 1.1 UPDATE

- Do not confuse with CONSTRUCT
- CONSTRUCT is an alternative for SELECT
- Instead of returning a table of result values, CONSTRUCT returns an RDF graph according to the template

SPARQL 1.1 UPDATE

- Do not confuse with CONSTRUCT
- CONSTRUCT is an alternative for SELECT
- Instead of returning a table of result values, CONSTRUCT returns an RDF graph according to the template
- SPARQL 1.1 UPDATE is a language to modify the given GRAPH
- `https://www.w3.org/TR/2013/REC-sparql11-update-20130321/`

SPARQL 1.1 UPDATE: Inserting triples

Inserting triples in a graph

```
INSERT DATA {
    tr:Bella dbp:name 'Bella' .
    ttr:Bella a tto:Cat .
    ttr:Ernesto a dbo:Person .
    ttr:Ernesto ttr:pet ttr:Bella .
}
```

A named `GRAPH` can be specified, otherwise the default graph is targeted.

SPARQL 1.1 UPDATE: Deleting triples

Inserting triples in a graph

```
DELETE DATA {
    tr:Bella dbp:name 'Bella' .
    ttr:Bella a tto:Cat .
    ttr:Ernesto a dbo:Person .
    ttr:Ernesto ttr:pet ttr:Bella .
}
```

A named `GRAPH` can be specified, otherwise the default graph is targeted.

SPARQL 1.1 UPDATE: Inserting/deleting conditionally

```
DELETE {
  ?x foaf:name ?y .
}
INSERT {
  ?x rdfs:label ?y .
}
WHERE {
  ?x foaf:name ?y .
  ?x a dbo:Person .
}
```

SPARQL 1.1 UPDATE: Deleting conditionally

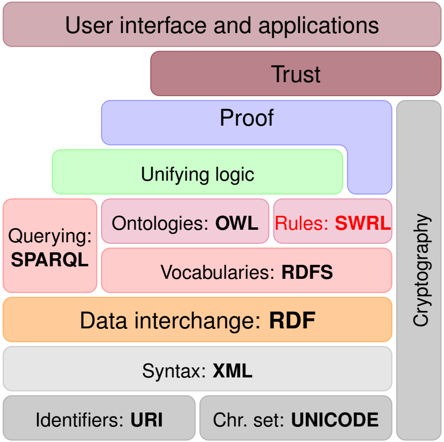
From specification:

Deleting old books

```
DELETE {
   ?book ?p ?v .
}
WHERE {
  ?book dc:date ?date .
  FILTER ( ?date < "2000-01-01T00:00:00"^^xsd:dateTime )
  ?book ?p ?v .
}
```

Reasoning with OWL and Rules

The Semantic Web Rule Language (SWRL)



The Semantic Web Rule Language (SWRL)

– Declarative language to define rules

– (Recall) The general form of an inference rule is:

$$\frac{P_1, \ldots, P_n}{P}$$

  – the P_i are **premises** (body), one or more;
  – and P is the **conclusion** (head), only one.

The Semantic Web Rule Language (SWRL)

- Declarative language to define rules
- (Recall) The general form of an inference rule is:

$$\frac{P_1, \ldots, P_n}{P}$$

  - the P_i are **premises** (body), one or more;
  - and P is the **conclusion** (head), only one.
- You can also find rules represented as
  - $P_1 \wedge \cdots \wedge P_n \rightarrow P$,
  - $P \leftarrow P_1 \wedge \cdots \wedge P_n$, or
  - P :- $P_1, \ldots, P_n$.

The Semantic Web Rule Language (SWRL)

- In our setting,
  - premises are **unary** or **binary** predicates (*i.e.*, triples); and
  - rules are **if-then statements** that add new triples to the graph.

The Semantic Web Rule Language (SWRL)

- In our setting,
  - premises are **unary** or **binary** predicates (*i.e.*, triples); and
  - rules are **if-then statements** that add new triples to the graph.
- SWRL can express things that are outside RDFS and OWL 2, *e.g.*,:

  `Customer(?c)` $\wedge$ `numberOfPizzasPurchased(?c, ?np)` $\wedge$
  `swrlb:greaterThan(?np, 1)` $\rightarrow$ `hasDiscount(?c, 0.2)`

The Semantic Web Rule Language (SWRL)

- In our setting,
  - premises are **unary** or **binary** predicates (*i.e.*, triples); and
  - rules are **if-then statements** that add new triples to the graph.
- SWRL can express things that are outside RDFS and OWL 2, *e.g.*,:
  $\texttt{Customer(?c)} \land \texttt{numberOfPizzasPurchased(?c, ?np)} \land \texttt{swrlb:greaterThan(?np, 1)} \rightarrow \texttt{hasDiscount(?c, 0.2)}$
- And also within OWL expressiveness:
  - Role chains: $\texttt{hasParent(?x1,?x2)} \land \texttt{hasBrother(?x2,?x3)} \rightarrow \texttt{hasUncle(?x1,?x3)}$
  - Subsumption: $\texttt{MeatPizza(?x)} \rightarrow \texttt{Pizza(?x)}$

The Semantic Web Rule Language (SWRL) vs SPARQL Update

SWRL rule:

$\texttt{foaf:name(?x, ?y)} \wedge \texttt{dbo:Person(?x)} \rightarrow \texttt{rdfs:label(?x, ?y)}$

SPARQL Update:

```
INSERT {
  ?x rdfs:label ?y .
}
WHERE {
  ?x foaf:name ?y .
  ?x a dbo:Person .
}
```

The Semantic Web Rule Language (SWRL) and Datalog

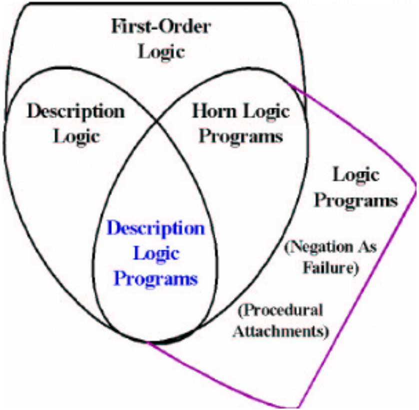
– **SWRL** was meant to combine the power of OWL and a subset of the Rule Markup Language (RuleML) $\rightarrow$ **union of both worlds**
  – RuleML is a subset of Datalog
  – Datalog is a declarative logic programming language
  – A Datalog program is composed of individual rules representing first-order logic **Horn clauses** (decidable fragment).

The Semantic Web Rule Language (SWRL) and Datalog

- **SWRL** was meant to combine the power of OWL and a subset of the Rule Markup Language (RuleML) → **union of both worlds**
  - RuleML is a subset of Datalog
  - Datalog is a declarative logic programming language
  - A Datalog program is composed of individual rules representing first-order logic **Horn clauses** (decidable fragment).
- The union of OWL and Rules brings undecidability:
  - Similar issue to OWL Entailment regimes for SPARQL.
  - Solution: restriction of the type of SWRL rules and OWL constructors.

“Complementing” the SWRL: DLPs, Datalog and OWL 2 RL

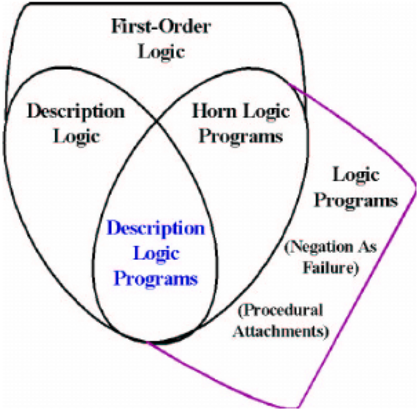
- **Description Logic Programs** (DLP): intersection between description logics and logic programs.
- **OWL 2 RL** has been inspired by DLPs.



Description Logic Programs: Combining Logic Programs with Description Logic. WWW 2003.

“Complementing” the SWRL: DLPs, Datalog and OWL 2 RL

- **Description Logic Programs** (DLP): intersection between description logics and logic programs.
- **OWL 2 RL** has been inspired by DLPs.
- OWL 2 RL axioms can be represented as **Datalog rules**.
- **Datalog engines** (like RDFox) can cope with **OWL 2 RL**, **SWRL** rules and other **Datalog rules** beyond DL and FOL.



Description Logic Programs: Combining Logic Programs with Description Logic. WWW 2003.

Data validation using (datalog) rules (i)

- **Ontologies** model the domain of application (e.g., expected cardinalities, relationships, accepted range of values for a *temperature sensor*).
- (Datalog) **Rules** to identify
  - *Missing data* (e.g., a person must have a name)
  - *Anomalies* according to the ontology (e.g., a person must have an age between 0 and 140)

Data validation using (datalog) rules (ii)

- Unlike databases, ontologies follow the **open world assumption**
- **Integrity constraints (IC)** are not supported by the Web Ontology Language (OWL):
  - *e.g., every person must have a name*

E. Kharlamov, B. Cuenca-Grau, E. Jiménez-Ruiz, et al. **Capturing Industrial Information Models with Ontologies and Constraints**. International Semantic Web Conference. 2016

Data validation using (datalog) rules (ii)

- Unlike databases, ontologies follow the **open world assumption**
- **Integrity constraints (IC)** are not supported by the Web Ontology Language (OWL):
  - *e.g., every person must have a name*
- Some **OWL axioms** can be **interpreted as ICs**
  - *e.g., cardinality restrictions (on the right hand side)*

E. Kharlamov, B. Cuenca-Grau, E. Jiménez-Ruiz, et al. **Capturing Industrial Information Models with Ontologies and Constraints**. International Semantic Web Conference. 2016

Data validation using (datalog) rules (ii)

- Unlike databases, ontologies follow the **open world assumption**
- **Integrity constraints (IC)** are not supported by the Web Ontology Language (OWL):
  - *e.g., every person must have a name*
- Some **OWL axioms** can be **interpreted as ICs**
  - *e.g., cardinality restrictions (on the right hand side)*
- ICs can be **captured with Datalog** (with negation as failure $\rightarrow$ **CWA**)
- This enables the use of **efficient datalog reasoners like RDFox**

E. Kharlamov, B. Cuenca-Grau, E. Jiménez-Ruiz, et al. **Capturing Industrial Information Models with Ontologies and Constraints**. International Semantic Web Conference. 2016

Data validation using (datalog) rules (iii)

- Project co-funded by **Siemens AG**: Modelling of complex industrial systems.
- Examples:
  - Turbine SUBCLASSOF TurboMachine AND (hasPart SOME Rotor)
  - TwoRotorTurbine SUBCLASSOF Turbine AND (hasPart EXACTLY 2 Rotor)
  - Turbine SUBCLASSOF hasPart ONLY Rotor

Data validation using (datalog) rules (iv)

OWL Axiom (α) interpreted as IC:

- Turbine SUBCLASSOF TurboMachine AND (hasPart SOME Rotor)

Missing information captured with a fresh Violation predicate.

- Resulting Datalog rules (**with negation as failure**):
  - Things_with_rotor(?x) $\leftarrow$ hasPart(?x, ?r) $\wedge$ Rotor(?r)
  - Violation(?t, α) $\leftarrow$ Turbine(?t) $\wedge$ **not** Things_with_rotor(?t)

Data validation using (datalog) rules (v)

OWL Axiom (α) interpreted as IC:

– MeatPizza SUBCLASSOF Pizza AND (hasIngredient SOME Meat)

Missing information captured with a fresh Violation predicate.

– Resulting Datalog rules as IC (**with negation as failure**):
  – Things_with_meat(?x) $\leftarrow$ hasIngredient(?x, ?m) $\wedge$ Meat(?m)
  – Violation(?p, α) $\leftarrow$ MeatPizza(?p) $\wedge$ **not** Things_with_meat(?p)

Data validation using (datalog) rules (vi)

OWL Axiom	Datalog rules
SubClassOf(A SomeValuesFrom(R B))	$R\_B(?x) \leftarrow R(?x, ?y) \ \wedge \ B(?y)$ and $Violation(?x, \alpha) \leftarrow A(?x) \ \wedge \ \textbf{not } R\_B(?x)$
SubClassOf(A HasValue(R b))	$Violation(?x, \alpha) \leftarrow A(?x) \ \wedge \ \textbf{not } R(?x, b)$
FunctionalProperty(R)	$R\_2(?x) \leftarrow R(?x, ?y_1) \ \wedge \ R(?x, ?y_2) \ \wedge$ $\textbf{not } owl{:}sameAs(?y_1, ?y_2)$ and $Violation(?x, \alpha) \leftarrow R\_2(?x)$
SubClassOf(A MaxCardinality(n R B))	$R\_(n{+}1)\_B(?x) \leftarrow \bigwedge_{1 \le i \le n+1} (R(?x, ?y_i) \wedge B(?y_i))$ $\bigwedge_{1 \le i < j \le n+1} (\textbf{not } owl{:}sameAs(?y_i, ?y_j))$ and $Violation(?x, \alpha) \leftarrow A(?x) \ \wedge \ R\_(n{+}1)\_B(?x)$
SubClassOf(A MinCardinality(n R B))	$R\_n\_B(?x) \leftarrow \bigwedge_{1 \le i \le n} (R(?x, ?y_i) \wedge B(?y_i))$ $\bigwedge_{1 \le i < j \le n} (\textbf{not } owl{:}sameAs(?y_i, ?y_j))$ and $Violation(?x, \alpha) \leftarrow A(?x) \ \wedge \ \textbf{not } R\_n\_B(?x)$

Capturing Industrial Information Models with Ontologies and Constraints. International Semantic Web Conference. 2016

SHACL: Shapes Constraint Language

SHACL: Shapes Constraint Language

- ✓ Standard W3C language to define constraints to validate RDF data.
- ✓ SHACL focus on CWA and provides a rich language to explicitly define checks over the data.
- ✗ Most (or all) of the features of SHACL could be mapped to DL, rules (*e.g.*, datalog) or SPARQL.
- ✓ Used in industry.
- ✓ Supported by database solutions like GraphDB and RDFox.

Shapes Constraint Language (SHACL): `https://www.w3.org/TR/shacl/`

SHACL VS Semantic Web Technology

– Is SHACL better than OWL? Better than rules?

– Is SHACL reinventing the wheel?

M. Andresei et al. Stable Model Semantics for Recursive SHACL, WWW 2020
B. Bogaerts et al. "SHACL: A Description Logic in Disguise", arXiv 2021
Expressiveness of B. Bogaerts et al. SHACL Features, ICDT 2022
M. Leinberger et al. Deciding SHACL Shape Containment through Description Logics Reasoning. ISWC 2020

SHACL Example (i)

Constraint	Description
minCount	Restricts minimum number of triples involving the focus node and a given predicate. Default value: 0
maxCount	Restricts maximum number of triples involving the focus node and a given predicate. If not defined = unbounded

```
:User a sh:NodeShape ;
   sh:property [
    sh:path      schema:follows ;
    sh:minCount 2 ;
    sh:maxCount 3 ;
  ] .
```

```
:alice schema:follows :bob,
                      :carol .

:bob   schema:follows :alice .   ☹

:carol schema:follows :alice,
                      :bob,       ☹
                      :carol,
                      :dave .
```

SHACL by example: `https://www.slideshare.net/jelabra/shacl-by-example`

SHACL Example (ii)

Constraint	Description
datatype	Restrict the datatype of all value nodes to a given value

```
:User a sh:NodeShape ;
   sh:property [
    sh:path      schema:birthDate ;
    sh:datatype xsd:date ;
  ] .
```

```
:alice schema:birthDate "1985-08-20"^^xsd:date .

:bob   schema:birthDate "Unknown"^^xsd:date .        ☹

:carol schema:birthDate 1990 .                       ☹
```

SHACL by example: `https://www.slideshare.net/jelabra/shacl-by-example`

SHACL Example (iii)

Constraint	Description
hasValue	Verifies that the focus node has a given value
in	Enumerates the value nodes that a property may have

```
:User a sh:NodeShape, rdfs:Class ;
  sh:property [
   sh:path       schema:affiliation ;
   sh:hasValue :OurCompany ;
  ];
  sh:property [
   sh:path schema:gender ;
   sh:in    (schema:Male schema:Female)
  ] .
```

```
:alice a :User;
       schema:affiliation :OurCompany ;
       schema:gender schema:Female .

:bob   a :User;
       schema:affiliation :AnotherCompany ;   ☹
       schema:gender schema:Male .

:carol a :User;
       schema:affiliation :OurCompany ;
       schema:gender schema:Unknown .          ☹
```

SHACL by example: `https://www.slideshare.net/jelabra/shacl-by-example`

SHACL Example (iv)

Constraint	Description
minInclusive	
maxInclusive	
minExclusive	
maxExclusive	

```
:Rating a sh:NodeShape ;
 sh:property [
   sh:path          schema:ratingValue ;
   sh:minInclusive 1 ;
   sh:maxInclusive 5 ;
   sh:datatype      xsd:integer
 ] .
```

```
:bad       schema:ratingValue 1 .
:average   schema:ratingValue 3 .
:veryGood  schema:ratingValue 5 .
:zero      schema:ratingValue 0 . ☹
```

SHACL by example: `https://www.slideshare.net/jelabra/shacl-by-example`

Graph Database Solutions

Introduction

Advanced graph database solutions:

- Scale to large Knowledge Graphs.
- Sophisticated indexing structures.
- Optimised reasoning.
- Fast query performance.
- Server solution in production.

State-of-the-art solutions

Dimensions:

- Free version.
- Compliance with Semantic Web standards.
- Reasoning capabilities.
- In-memory or In-disk.
- Documentation and installation requirements.
- Additional features.

A Survey of RDF Stores & SPARQL Engines for Querying Knowledge Graphs. arXiv:2102.13027 2021 (Appendix A)

Semantic Web standards

- The **World Wide Web Consortium (W3C)** is an international community that develops open standards to ensure the long-term growth of the Web: `https://www.w3.org/`
- On the Web and **beyond**.
- **Why standards?**
  - broader industry (and academic) agreement,
  - interoperability across organizations and applications,
  - avoids vendor lock-in of a particular (exchange or query) format.

Apache Jena TDB

– Free solution.

– Provides a native (in-disk) RDF store.

– In combination with Jena Fuseki to provide SPARQL Endpoint support.

✗ Supports reasoning as in Jena, but not direct support for OWL 2 nor the OWL 2 profiles.

`https://jena.apache.org/documentation/tdb/`

OpenLink Virtuoso

✓ Provides the SPARQL endpoint for DBpedia.

– Open source and commercial versions.

– Object-oriented database model.

✓ Native graph model storage provider for Jena and RDF4J.

✗ Custom inference rules. Partial support for OWL 2.

`https://virtuoso.openlinksw.com/`

`http://vos.openlinksw.com/owiki/wiki/VOS`

Blazegraph

- ✓ Provides the SPARQL Endpoint for Wikidata.
- ✓ Free and open source.
- – Both in-memory and disk-oriented storage.
- ✗ Only supports OWL 1 Lite reasoning.
- – Blazegraph team now working for Amazon.

`https://blazegraph.com/`

AllegroGraph

– Free and commercial licenses.

✓ Support for OWL 2 RL materialization.

✓ Client interface in several languages.

– Can be used to query both documents and graph data (via SPARQL).

`https://allegrograph.com/`

Neo4j

✓ Open source graph database.

– Based on the Property Graph Model.

✗ Cypher as graph query language (no native SPARQL support).

– Support *via a plugin* for RDF, RDFS and OWL vocabularies.

✗ Basic inferencing support.

✓ Support for Analytics.

✓ Interfaces in many languages.

`https://neo4j.com/`

TypeDB (formerly GRAKN.AI)

✓ TypeDB is an open-source, distributed knowledge graph database.

✓ Support for analytics.

✓ Interesting integration with machine learning models.

– Provides inferencing support.

✗ No support for any of the Semantic Web standards.

`https://vaticle.com/`
GRAKN.AI vs Semantic Web standards (some justifications that I do not personally share):
`https://blog.grakn.ai/knowledge-graph-representation-grakn-ai-or-owl-506065bd3f24`

GraphDB (formerly OWLIM)

- – Free and commercial versions.
- ✓ Very easy to install and use.
- ✓ Powerful reasoning features: including OWL 2 QL and RL profiles.
- ✓ Supports SHACL validation.
- – Includes text indexing via lucene.
- – Powered the early Linked Data services at the BBC.
- – **Our choice for the lab today.**

`https://www.ontotext.com/products/graphdb/`

RDF4J (formerly Sesame)

✓ General Java framework to manage RDF data.

– Provides native (in-memory and in-disk) storage solutions.

– Can connect to other RDF stores *e.g.*,: Blazegraph, Amazon Neptune, GraphDB, Virtuoso.

– Indexing, reasoning and query processing techniques depend on the underlying storage engine.

`https://rdf4j.org/`

RDFox

– Commercial system. Free academic license on request.

✓ Support for materialization-based datalog reasoning (including OWL 2 RL and SWRL rules).

✓ Supports SHACL validation.

– In-memory RDF engine.

✓ Access via Java API or remotely via REST API or SPARQL Endpoint.

✗ Limitation on the size of the memory.

`https://www.oxfordsemantic.tech/product`

Amazon Neptune

– Cloud-based only solution.

– Blazegraph is now part of Amazon Neptune.

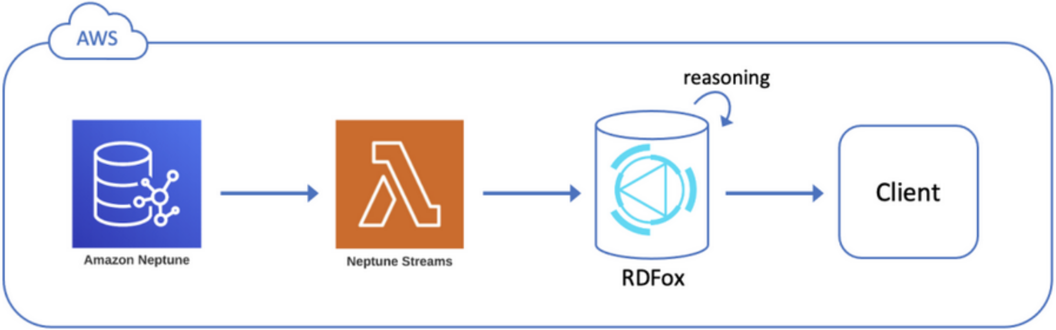
✗ On-Demand pricing.

✗ Native inferencing is not yet supported.

✓ Keep an eye on future development!

`https://aws.amazon.com/neptune/`

Amazon Neptune + RDFox



Amazon Neptune and RDFox: `https://aws.amazon.com/blogs/database/use-semantic-reasoning-to-infer-new-facts-from-your-rdf-graph-by-integrating-rdfox-with-amazon-neptune/`

Graph database/Triplestore benchmarking

- Oracle Database 12c: 1.08 Trillion triples
- AnzoGraph DB: 1.06 Trillion triples
- AllegroGraph: 1.0 Trillion triples
- Virtuoso: 94.2 Billion Triples
- Stardog: 50 Billion triples
- **RDFox**: 19.47 Billion triples
- **GraphDB**: 17 Billion triples
- Apache Jena TBD: 16.7 billion triples.

Numbers may not be up-to-date: `https://www.w3.org/wiki/LargeTripleStores`

Laboratory

Laboratory

- SPARQL 1.1 queries and SPARQL 1.1 Update
- Using the RDF store GraphDB
  - Creating repositories.
  - Loading data and ontology.
  - Querying the data.
  - Named graphs
- A bit of SHACL and SWRL from `Protégé` (Pizza tutorial chapters 10 and 11)