Advanced RDF(S)

Michal Med

November 5, 2020

1 Introduction

Today, we explore two more advanced features of the RDF(S) stack:

- RDFS reasoning – to infer new knowledge, and
- RDF validation – to check RDF data w.r.t. some constraints

Remark: In this document, we use the following prefixes:

```
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#> .
@prefix :  <http://onto.fel.cvut.cz/ontologies/shacl-example/> .
@prefix sh:  <http://www.w3.org/ns/shacl#> .
@prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix skos:  <http://www.w3.org/2004/02/skos/core#> .
@prefix dc:  <http://purl.org/dc/terms/> .
```

2 RDF(S) Reasoning

RDF(S) is used to describe data schemas and infer new knowledge. Basic knowledge about reasoning strategies and its implementation in GraphDB can be found in relevant references. In GraphDB, RDFS inference is performed using *forward chaining*. Given an RDF graph G (e.g. the content of a GraphDB repository), the reasoner takes original RDF triples $\{ot_i\} = G$ and a set of *inference rules* to generate new RDF triples $\{nt_i\}$. Then, the given SPARQL query is evaluated on $G' = \{ot_i\} \cup \{nt_i\}$. In GraphDB **the reasoner is executed upon data insert/update**.

2.1 Exercises

In the new repositories called [USER]–osw2020 is enabled reasoning. Insert the following snippet into the new named graph:

```
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix : <http://onto.fel.cvut.cz/ontologies/__repository__/>.

:John :hasWife :Sue .
:John a :Man .
:hasWife rdfs:domain :MarriedMan ;
         rdfs:range  :MarriedWoman ;
         rdfs:subPropertyOf :hasRelative .
```

You can add subgraphs as new named graphs to see the results.

Ex. 1 — Does the reasoner tell something about :John being :MarriedMan using RDFS entailment on the imported data? Use ASK statement to answer.

Answer (Ex. 1) — John is a married man.

```
ASK { :John a :MarriedMan .}
```

Ex. 2 — What new triples about :Sue are added as the output of a reasoner using RDFS entailment on the RDF Snippet:

```
:John :hasRelative :Sue .
:hasWife rdfs:subPropertyOf :hasRelative .
:hasWife rdfs:range :MarriedWoman .
```

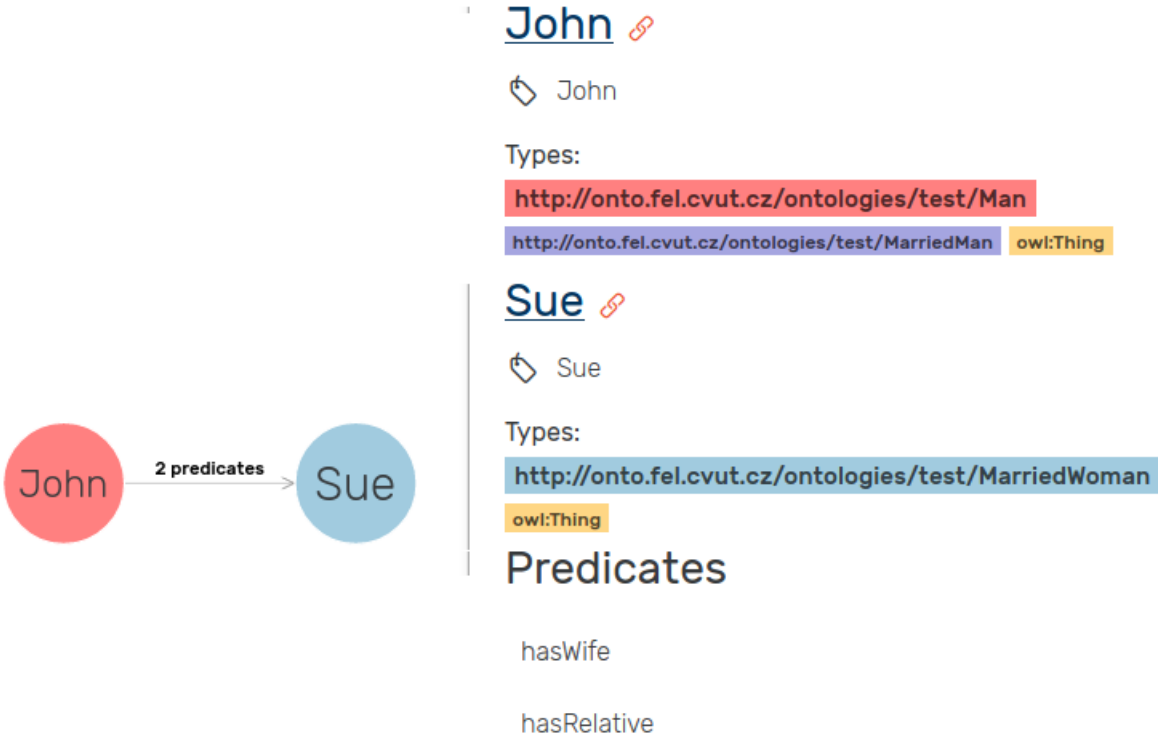
Verify answer in the Graphs overview.

Answer (Ex. 2) — The reasoner entails no new triple about `:Sue`. Two rows shown in the picture are not caused by the RDFS entailment.

	subject	predicate	object	context
1	:Sue	rdf:type	owl:Thing	http://www.ontotext.com/implicit
2	:Sue	owl:sameAs	:Sue	http://www.ontotext.com/implicit

Ex. 3 — What new information (trilpes) about :Sue and :John are added as the output of a reasoner using RDFS entailment on the imported data? Verify answer using Visual graph.

Answer (Ex. 3) — The reasoner entails several new triples: `:Sue a :MarriedWoman`, `:John a :MarriedMan` and `:John :hasRelative :Sue`.
However, validation capabilities of RDFS are limited. How to check that each `:MarriedMan` has exactly one `:hasWife` property ?



3 Intro to SHACL

SHACL[1] is a W3C recommendation aiming at validation of RDF data using so called *shapes*. Shapes are expressed in RDF, e.g.:

```
:MarriedManShape
    a sh:NodeShape ;
    sh:targetClass :MarriedMan ;
    sh:property [
        sh:path :hasWife ;
        sh:minCount 1 ;
        sh:maxCount 1 ;
    ] .
```

Shapes are class-centric. Here, we define a shape for the RDFS class `:MarriedMan`. This shape checks that each instance of this class is explicitely related to exactly one other instance through the property `:hasWife`. Validating the RDF snippet

```
:John a :MarriedMan .
```

[1] https://www.w3.org/TR/shacl/

against the shape produces a validation error, as `:John` has no explicitely stated `:hasWife` relation, while validating the RDF snippet

```
:John a :MarriedMan ;
      :hasWife :Sue .
```

against the shape passes. You can test both examples e.g. at `http://shacl.org/playground/`.

Refer to the SHACL specification `https://www.w3.org/TR/shacl/` for details and other constructs

3.1 Exercises

Ex. 4 — The following SHACL example states the basic rules of SKOS, as it is created by VocBench. Therefore, constraints can be used for validating SKOS vocabulary you created in the previous seminar. You shall be able to access repository `osw2020-history-[groupnumber]-core` with your account. Download the graph with `history` data and paste it into SHACL playground. If you try to validate it, you encounetr several errors. They are caused by two bugs in the SHACL constraints. Find them and fix the, Once you fix them correctly, validation shall pass.

```
@prefix dash: <http://datashapes.org/dash#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix schema: <http://schema.org/> .
@prefix sh: <http://www.w3.org/ns/shacl#> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
@prefix skos: <http://www.w3.org/2004/02/skos/core#> .
@prefix dc: <http://purl.org/dc/terms/> .

:SkosConceptShape
    a sh:NodeShape ;
    sh:targetClass skos:Concept ;
    sh:property [
        sh:path skos:inScheme ;
        sh:minCount 1 ;
    ] ;
    sh:property [
        sh:path dc:created ;
        sh:minCount 1 ;
        sh:datatype xsd:dateTime
    ] ;
    sh:property [
        sh:path skos:inScheme ;
        sh:minCount 1 ;
        sh:class skos:Conceptscheme
```

```
    ] ;
    sh:property [
        sh:path skos:prefLabel ;
        sh:minCount 1 ;
    ] .

:SkosConceptSchemeShape
    a sh:NodeShape ;
    sh:targetClass skos:ConceptScheme ;
    sh:property [
        sh:path skos:prefLabel ;
        sh:minCount 2 ;
    ] .
```

Answer (Ex. 4) — The part checking if `Concept` is `inScheme` `ConceptScheme` has a typo: in `ConceptScheme` shall be capital `S`:

```
    sh:property [
        sh:path skos:inScheme ;
        sh:minCount 1 ;
        sh:class skos:ConceptScheme
    ] ;
```

The part checking `ConceptScheme` label requires two `prefLabels`. It shall require only one:

```
:SkosConceptSchemeShape
    a sh:NodeShape ;
    sh:targetClass skos:ConceptScheme ;
    sh:property [
        sh:path skos:prefLabel ;
        sh:minCount 1 ;
    ] .
```

Ex. 5 — Extend the corrected shapes to require each `skos:ConceptScheme` to have at least two `skos:hasTopConcept` relations.

Answer (Ex. 5) — The corrected SHACL shapes are as follows:

```
schema:SkosConceptSchemeShape
    a sh:NodeShape ;
    sh:targetClass skos:ConceptScheme ;
    sh:property [
        sh:path skos:prefLabel ;
        sh:minCount 1 ;
    ];
    sh:property [
```

```
        sh:path skos:hasTopConcept ;
        sh:minCount 2 ;
    ] .
```

Ex. 6 — Use the extended version of shapes to validate your SKOS vocabulary from the previous section. Correct the failing data in your vocabulary, if any.

Ex. 7 — **Bonus point** *Design a simple SHACL specification for FOAF.* In one of the previous seminars, you had to create your FOAF profile. Consider the 5 most important properties of FOAF, create a SHACL specification for them only and validate your FOAF profile. Upload the FOAF and SHACL to your gitlab repo and send a `.txt` with commit to **BRUTE**.

4 RDFS Entailment Rules

RDFS-entailment w.r.t D interprets most RDF and RDFS vocabulary. In the table below, G denotes an RDF graph, D denotes the set of datatypes.

rule	G contains	t_i, s.t. $G \models_{RDFS-D} t_i$
rdfs1	any IRI $dIRI \in D$ in G	$(dIRI, \texttt{rdf:type}, \texttt{rdfs:Datatype})$
rdfs2	(s,p,o), $(p, \texttt{rdfs:domain}, w)$	$(s, \texttt{rdf:type}, w)$
rdfs3	(s,p,o), $(p, \texttt{rdfs:range}, w)$	$(o, \texttt{rdf:type}, w)$
rdfs4	(s,p,o)	$(s, \texttt{rdf:type}, \texttt{rdfs:Resource})$ $(o, \texttt{rdf:type}, \texttt{rdfs:Resource})$
rdfs5	$(p_1, \texttt{rdfs:subPropertyOf}, p_2)$ $(p_2, \texttt{rdfs:subPropertyOf}, p_3)$	$(p_1, \texttt{rdfs:subPropertyOf}, p_3)$
rdfs6	$(p, \texttt{rdf:type}, \texttt{rdf:Property})$	$(p, \texttt{rdfs:subPropertyOf}, p)$
rdfs7	$(p_1, \texttt{rdfs:subPropertyOf}, p_2)$ (s, p_1, o)	(s, p_2, o)
rdfs8	$(s, \texttt{rdf:type}, \texttt{rdfs:Class})$	$(s, \texttt{rdfs:subClassOf}, \texttt{rdfs:Resource})$
rdfs9	$(c_1, \texttt{rdfs:subClassOf}, c_2)$ $(s, \texttt{rdf:type}, c_1)$	$(s, \texttt{rdf:type}, c_2)$
rdfs10	$(c, \texttt{rdf:type}, \texttt{rdfs:Class})$	$(c, \texttt{rdfs:subClassOf}, c)$
rdfs11	$(c_1, \texttt{rdfs:subClassOf}, c_2)$ $(c_2, \texttt{rdfs:subClassOf}, c_3)$	$(c_1, \texttt{rdfs:subClassOf}, c_3)$
rdfs12	$(p, \texttt{rdf:type},$ $\texttt{rdfs:ContainerMembershipProperty})$	$(p, \texttt{rdfs:subPropertyOf},$ $\texttt{rdfs:member})$
rdfs13	$(d, \texttt{rdf:type}, \texttt{rdfs:Datatype})$	$(d, \texttt{rdfs:subClassOf}, \texttt{rdfs:Literal})$

5 Relevant References

- Reasoning strategies `https://graphdb.ontotext.com/documentation/9.1/standard/introduction-to-semantic-web.html#introduction-to-semantic-web`

- GraphDB reasoning https://graphdb.ontotext.com/documentation/9.1/standard/reasoning.html,
- R-entailment https://www.semanticscholar.org/paper/Combining-RDF-and-Part-of-3A-Horst/09f747b35d0e819baab202593476723c8c19d571,
- OWL compliance https://graphdb.ontotext.com/documentation/9.1/standard/owl-compliance.html
- RDFS-plus rules https://docs.cambridgesemantics.com/anzograph/v2.2/userdoc/inferences.htm