Advanced RDF(S)

Petr Křemen

November 8, 2018

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. 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

Ex. 1 — What is the output of a reasoner using RDFS entailment on the RDF Snippet:

```
:John :hasWife :Sue .
:John a :Man .
:hasWife rdfs:domain :MarriedMan .
```

Ex. 2 — What is the output of a reasoner using RDFS entailment on the RDF Snippet:

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

Ex. 3 — What is the output of a reasoner using RDFS entailment on the RDF Snippet:

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

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 .
```

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

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

3.1 Exercises

Ex. 4 — The following SHACL constraints can be used for validating SKOS vocabulary you created in the previous seminar. However, there are two bugs in the shapes, find them and correct them. When you use the corrected version for validation of the the `s5-animals-6` repository content (download it and paste it into the SHACL playground), it should pass.

```
: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 1 ;
    ] .
```

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

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

Ex. 7 — *Design a simple SHACL specification for FOAF – consider the 5 most importat properties only and validate your FOAF profile .*

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	$(d, \texttt{rdf:type}, \texttt{rdfs:Datatype})$	$(d, \texttt{rdfs:subClassOf}, \texttt{rdfs:Literal})$