SHACL2SPARQL: Validating a SPARQL Endpoint against Recursive SHACL Constraints

Julien Corman[1], Fernando Florenzano[2], Juan L. Reutter[2], and Ognjen Savković[1]

[1] Free University of Bozen-Bolzano, Bolzano, Italy
[2] PUC Chile and IMFD, Chile

Abstract. The article presents SHACL2SPARQL, a tool that validates an RDF graph stored as a SPARQL endpoint against possibly recursive SHACL constraints. It is based on the algorithm proposed in [3]. This implementation improves upon the original algorithm with a wider range of natively supported constraint operators, SPARQL query optimization techniques, and a mechanism to explain invalid targets.

1 Introduction

RDF shapes have recently emerged as an intuitive way to formulate constraints over RDF graphs. In particular, SHACL (for SHApe Constraint Language) has become a W3C recommendation in July 2017. A SHACL *schema* is composed of so-called *shapes*, which define properties to be verified by a node and its neighbors. As an illustration, Figure 1 contains two SHACL shapes. Shape `DirectorShape` states that a director must have an IMDB identifier, whereas shape `MovieShape` states that a movie must have at least one director, and that all of them must verify `DirectorShape`. In addition, `MovieShape` defines its *targets*, namely all instances of the class `Movie`, which must be validated against this shape. `DirectorShape` on the other hand has no target definition.

A key feature of shape-based constraints is the possibility for a shape to refer to another, like `MovieShape` refers to `DirectorShape` in Figure 1. This allows designing schemas in a modular fashion, but also reusing existing shapes in a new schema, thus favoring semantic interoperability. However, validation techniques for schemas with references are still at an early stage. In particular, the SHACL specification leaves explicitly undefined the semantics of so-called *recursive* schemas, i.e. schemas where some shape refers to itself, either directly or via a reference cycle.

Another important aspect of shape-based validation is the storage of the graph. RDF datasets are often exposed as endpoints, and accessed via SPARQL queries. Validating a non-recursive schema in this context is relatively straightforward, since non-recursive SHACL schemas (even with references) have a natural translation to SPARQL. But recursive schemas exceed the expressive power of SPARQL, making validation more involved: if the schema is recursive, it is not possible in general to retrieve all nodes violating a given shape by executing a

```
:MovieShape
  a sh:NodeShape ;                :DirectorShape
  sh:targetClass dbo:Film ;         a sh:NodeShape ;
  sh:property [                     sh:property [
    sh:path dbo:director ;            sh:path dbo:ImdbId ;
    sh:node :DirectorShape ;          sh:minCount 1 ] .
    sh:minCount 1 ] .
```

Fig. 1: Two SHACL shapes, about movies and directors

single SPARQL query over the endpoint. Therefore some extra computation is required in order to validate an endpoint against such schema.

This opens a wide range of possibilities. At one end of the spectrum, validation may be performed in-memory, relying on simple SPARQL queries, whose purpose is to retrieve subgraphs to be validated. At the other end of the spectrum, validation may be delegated as much as possible to query evaluation, but queries in this case may be complex and/or partially redundant. The right compromise may be dictated by available resources (e.g. which proportion of the graph can effectively be loaded into memory), and/or the performance of the endpoint for a given type of queries (depending on physical storage, indexes, parallelization, etc.).

An abstract algorithm was proposed in [3], for two recursive but tractable fragments of SHACL. The approach favors a limited number of simple queries (one per shape) with low selectivity, whereas the rest of the validation process is handled by some in-memory rule-based inference.

The current paper presents a prototype implementation of this algorithm, named SHACL2SPARQL, which is the first implementation (to our knowledge) of recursive SHACL validation over an endpoint. It improves upon the abstract algorithm thanks to *(i)* a wider range of natively supported constraint operators, *(ii)* a mechanism to explain invalid targets, *(iii)* query optimization techniques. The source code is available online [2], together with execution and building instructions.

Due to space limitations, the article only sketches the original algorithm, and focuses on SHACL2SPARQL functionalities, in particular how it improves upon [3].

2 Validation Algorithm

The algorithm defined in [3] takes as input an RDF graph and a shape schema in normal form, i.e. whose constraints use only a small set of core operators (conjunction, negation, minimum cardinality, etc.), and no nested operator. A normalized schema can in turn be obtained in linear time from a non-normalized one, by introducing fresh shapes for sub-expressions. The validation algorithm processes each shape s in this normalized schema, one after the other. A SPARQL query $q_{\text{def}(s)}$ is generated for s, which retrieves each node v that may verify the constraints for s, as well as the neighbors of v needed to validate it against s. For instance, for shape `MovieShape` in Figure 1:

$$q_{\text{def}(\texttt{MovieShape})} = \texttt{SELECT ?x ?y WHERE \{?x dbo:director ?y\}}$$

This query retrieves all nodes (bound to `?x`) that have a `dbo:director`-successor, and thus may verify the constraints for `MovieShape`. It also retrieves these `dbo:director`-successors (bound to `?y`), which will in turn be validated against shape `DirectorShape`.

The answers to $q_{\mathrm{def}(s)}$ over the graph are then used to generate a set of Boolean formulas, which may be viewed as rules, one per answer. These rules are added to the set of rules computed so far, and some in-memory inference is performed, identifying all targets that can be inferred to be either valid or violated at this stage, and discarding unnecessary rules. After this inference phase, another shape is selected, and the process repeated until either all shapes have been processed, or each target has been validated or violated. The specific order in which shapes are processed does not affect soundness.

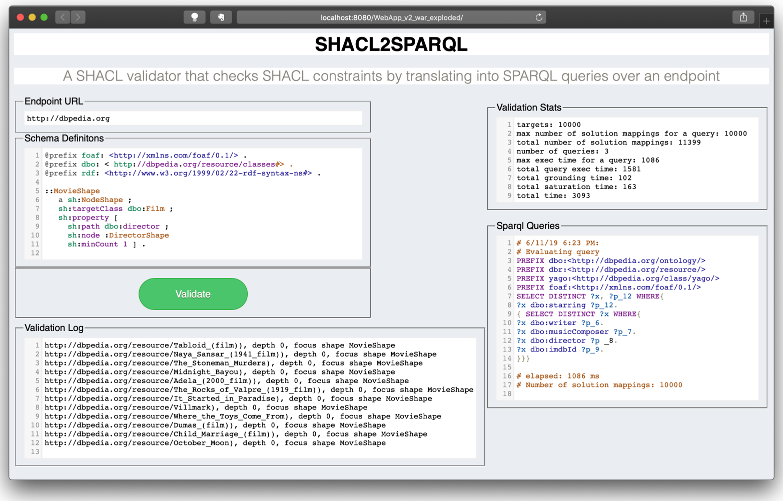
3 SHACL2SPARQL

SHACL2SPARQL takes as input (the address of) a SPARQL endpoint and a SHACL schema. A first improvement over[3] is the format of the schema.

The algorithm executed by SHACL2SPARQL natively handles constraints with additional operators (disjunction, maximum cardinality, "for all", etc.), although it cannot yet handle the full SHACL syntax. An extension to non-normalized schemas within the two tractable fragments is under development. This does not increase the expressive power of constraints (these extra operators can be expressed by combining core operators), but may reduce the number of queries to be evaluated against the endpoint (one per shape). SHACL2SPARQL supports two alternative concrete syntaxes for constraints, either the RDF (turtle) syntax of the SHACL specification (parsed with the Shaclex [1] library), or an ad-hoc (and more user-friendly) JSON syntax.

Another improvement pertains to traceability. The shape ordering strategy followed by SHACL2SPARQL consists in prioritizing shapes with a target definition, then the shapes that they refer to (if not processed yet), etc. As a side effect, the validation report produced by SHACL2SPARQL contains possible explanations for the failure of each invalid target, in the form of a sequence of shape references. For instance, let s_0 be a shape with non-empty target definition. And let us assume three additional shapes s_1, s_2, s_3 with empty target definitions, and such that s_0 refers to s_1 and s_3, s_1 refers s_2, and s_2 refers to s_3. Then the shape processing order (from left to right) is $[\{s_0\}, \{s_1, s_3\}, \{s_2\}]$ (where $\{s_i, s_j\}$ means that s_i and s_j can be processed in any order). Now let v be a target node for s_0, and let us assume that this target is inferred to be violated immediately after processing shape s_3. This must be due to a constraint in s_0 referring to shape s_3 (like `MovieShape` refers to `DirectorShape` in Figure 1). So the sequence $s_0 \rightarrow s_3$ is returned as an explanation. Other explanations are possible, but their number is worst-case exponential, which is why SHACL2SPARQL only returns the first one found. This strategy also guarantees that (one of) the *shortest* explanations is returned (e.g. $s_0 \rightarrow s_1 \rightarrow s_2 \rightarrow s_3$ may be another in this example).

As additional output, SHACL2SPARQL provides the generated SPARQL queries, their evaluation order, and various statistics about the validation process (execution times, number of answers to each query, number of valid and violated targets after each inference phase, etc.).



SHACL2SPARQL

A SHACL validator that checks SHACL constraints by translating into SPARQL queries over an endpoint

Endpoint URL

```
http://dbpedia.org
```

Schema Definitons

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix dbo: < http://dbpedia.org/resource/classes#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .

::MovieShape
   a sh:NodeShape ;
   sh:targetClass dbo:Film ;
   sh:property [
     sh:path dbo:director ;
     sh:node :DirectorShape
     sh:minCount 1 ] .
```

Validation Log

```
http://dbpedia.org/resource/Tabloid_(film)), depth 0, focus shape MovieShape
http://dbpedia.org/resource/Naya_Sansar_(1941_film)), depth 0, focus shape MovieShape
http://dbpedia.org/resource/The_Stoneman_Murders), depth 0, focus shape MovieShape
http://dbpedia.org/resource/Midnight_Bayou), depth 0, focus shape MovieShape
http://dbpedia.org/resource/Adela_(2000_film)), depth 0, focus shape MovieShape
http://dbpedia.org/resource/The_Rocks_of_Valpre_(1919_film)), depth 0, focus shape MovieShape
http://dbpedia.org/resource/It_Started_in_Paradise), depth 0, focus shape MovieShape
http://dbpedia.org/resource/Villmark), depth 0, focus shape MovieShape
http://dbpedia.org/resource/Where_the_Toys_Come_From), depth 0, focus shape MovieShape
http://dbpedia.org/resource/Dumas_(film)), depth 0, focus shape MovieShape
http://dbpedia.org/resource/Child_Marriage_(film)), depth 0, focus shape MovieShape
http://dbpedia.org/resource/October_Moon), depth 0, focus shape MovieShape
```

Validation Stats

```
targets: 10000
max number of solution mappings for a query: 10000
total number of solution mappings: 11399
number of queries: 3
max exec time for a query: 1086
total query exec time: 1581
total grounding time: 102
total saturation time: 163
total time: 3093
```

Sparql Queries

```
# 6/11/19 6:23 PM:
# Evaluating query
PREFIX dbo:<http://dbpedia.org/ontology/>
PREFIX dbr:<http://dbpedia.org/resource/>
PREFIX yago:<http://dbpedia.org/class/yago/>
PREFIX foaf:<http://xmlns.com/foaf/0.1/>
SELECT DISTINCT ?x, ?p_12 WHERE{
?x dbo:starring ?p_12.
{ SELECT DISTINCT ?x WHERE{
?x dbo:writer ?p_6.
?x dbo:musicComposer ?p_7.
?x dbo:director ?p _8.
?x dbo:imdbId ?p_9.
}}}

# elapsed: 1086 ms
# Number of solution mappings: 10000
```

Fig. 2: Screenshot of the web interface

Other improvements pertains to query optimization. For instance, a potential bottleneck of the abstract algorithm defined in [3] is shapes with cardinality constraints, e.g. "`sh:minCount` k". If $k > 1$, and if a node v has n r-successors with $n > k$, then $q_{\text{def}(s)}$ may have $\binom{n}{k}$ answers for v and its immediate neighbors. To avoid such combinatorial explosion, it is possible in some cases to reduce the selectivity of $q_{\text{def}(s)}$, by extending it with a nested subquery that acts as a filter. If no other shape refers to s, then this subquery is the target definition of s. Or if only one other shape s' refers to s, and if s has an empty target definition, then this subquery is the optimized query (defined inductively) for s'.

4 Demo and Web Interface

The demonstration will offer the possibility to test SHACL2SPARQL via a web interface, a screenshot of which is shown in Figure 2. A connection to a local instance of DBPedia will be provided, together with predefined SHACL schemas. Attendees will be offered the possibility to write their own shapes, analyze the output of the validation process (explanations for invalid targets), understand the execution of the algorithm (visualizing SPARQL queries), and appreciate the performance of the tool thanks to statistics (e.g. time spent evaluating queries vs time spent reasoning).

Acknowledgments. This work was supported by the QUADRO, ROBAST, OBATS and ADVANCE4KG projects at the Free University of Bozen-Bolzano, and the Millennium Institute for Foundational Research on Data (IMFD), Chile.

References

[1] Shaclex. `github.com/labra/shaclex`.
[2] Shacl/sparql, source code. `github.com/rdfshapes/shacl-sparql`.
[3] J. Corman, F. Florenzano, J. L. Reutter, and O. Savković. Validating SHACL constraints over a SPARQL endpoint. In *ISWC*, 2019.