Towards a Robust Semantics for SHACL: Preliminary Discussion

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

[1] Free University of Bozen-Bolzano, Bolzano, Italy
[2] PUC Chile and Center for Semantic Web Research, Santiago, Chile

Abstract. Validating RDF graphs against constraints has gained interest in recent years, due to the popularity of RDF and the growth of knowledge bases. SHACL, a constraint language for RDF, has recently become a W3C recommendation, with a specification detailing syntax, semantics and common use cases. Unfortunately, this (otherwise complete) specification does not cover validation against recursive constraints. This omission is important, because SHACL by design favors constraint references. We investigate the possibility of a formal semantics for SHACL which covers the recursive case, while being compliant with the current standard.

1 Introduction

Recent years have seen an increased interest in standardizing a declarative constraint language for RDF, and devising mechanisms to detect violations of such constraints. One of the most promising candidate languages is SHACL, or Shapes Constraint Language,[3] which has become a W3C recommendation in 2017.

SHACL constraints are grouped in so-called "shapes" to be verified by certain nodes of the graph under validation, and such that a shape may reference another. Figure 1 presents two SHACL shapes. The left one, named `:AddressShape`, is meant to define valid addresses. A node v satisfying this shape must satisfy two (nested) constraints: the first one states that there must be at least one successor (`sh:minCount 1`) of v via property `:telephone`, and the second one states that there must be at most one successor of v via property `:postalCode`.

Validating an RDF graph against a set of shapes is based on the notion of "target nodes", which mandates for each shape which nodes have to conform to it. In Figure 1, the right shape contains the triple `:PersonShape sh:targetClass :Person`, stating that its target nodes are all nodes of type `:Person`. Nodes may have to conform to additional shapes, due to shape references. For example, the right shape of Figure 1 contains one (non-recursive) shape reference, stating that every node v conforming to `:PersonShape` has at most one `:address`, which must conform to `:AddressShape`, and one recursive reference, stating that each `foaf:friend` of v must conform to `:PersonShape`. By *recursion*, we refer to (possibly n-ary) shape reference cycles. Unfortunately, the validation of a graph

[3] `https://www.w3.org/TR/shacl`

```
:AddressShape
  a sh:NodeShape ;
  sh:property [
    sh:path :telephone ;
    sh:minCount 1
  ] ;
  sh:property [
    sh:path :postalCode ;
    sh:maxCount 1
  ] .

:PersonShape
  a sh:NodeShape ;
  sh:targetClass :Person ;
  sh:property [
    sh:path :address ;
    sh:maxCount 1 ;
    sh:node :AddressShape
  ] ;
  sh:property [
    sh:path foaf:friend ;
    sh:node :PersonShape
  ] .
```

Fig. 1. Two SHACL shapes, about persons and addresses

against recursive shapes is left explicitly undefined in the SHACL specification. This is an important limitation for RDF graphs which are not tree-shaped. As an illustration, in Example 1, the RDF graph G_1 is valid w.r.t. the shapes of Figure 1, whereas G_2 is invalid, and the validity of G_3 is left unspecified.

Example 1.
$G_1 = \{$`:p1 rdf:type :Person ; :address :a2 . :a2 :telephone :t3 .`$\}$
$G_2 = G_1 \cup \{$`:a2 :postalcode :c4 . :a2 :postalcode :c5 .`$\}$
$G_3 = G_1 \cup \{$`:p1 foaf:friend :p6 . :p6 foaf:friend :p1 .`$\}$

In this work, we set up the task of defining a robust semantics for graph validation in SHACL (including circular references), and studying its computational properties. A report is available online, which reflects the current state of our investigations: `https://www.inf.unibz.it/krdb/KRDB%20files/tech-reports/KRDB18-01.pdf`.

Related Work. Boneva et al. [2, 7, 1, 3] have embarked on a study of a similar constraint language for RDF, called Shape Expression Language, or ShEx,[4] providing in [3] an abstract syntax and semantics. A different abstract language in provided in [1], which encodes a common fragment of ShEx and SHACL. In both cases, validating a graph against a recursive set of shapes is based on the existence of a *typing* (of nodes with shape names) verifying constraints and targets, which is also the approach we follow (with the notion of *shape assignment* described below). Perhaps the most important difference with our investigations is that these two studies are restricted to constraints with stratified negation (in the datalog sense), whereas we consider constraints with arbitrary negation. Another key difference (in the stratified case) is that the semantics given in [3] relies on a specific maximal typing (defined inductively on the constraint's strata), which is not the only possible one. We hope that both works may complement each other, towards an optimal constraint language for RDF.

Another related line of work, advocated in [5] as a semantic grounding for SHACL, is inspired by Description Logics under closed-world assumption [8], effectively reducing graph validation to FO satisfiability with closed (binary) predicates [6]. But as illustrated with Example 2, (2-valued) FO satisfiability may not be well-suited for validating target nodes against non-stratified constraints.

[4] `http://shex.io/shex-semantics`

2 Validating a graph against recursive constraints

In order to devise a formal semantics, we abstract away from the concrete syntax of SHACL. For shape constraints, we chose a concise notation inspired by Description Logics, expressive enough to encode the SHACL "Core Constraint Components" of the specification. For instance, the constraints in Figure 1 would be translated into two constraint definitions, one for each shape:

$$\begin{aligned}\text{AddressShape} &\doteq (\geq_1 \text{telephone}.\top) \wedge (\leq_1 \text{postalCode}.\top)\\ \text{PersonShape} &\doteq (\leq_1 \text{address}.\top) \wedge (\leq_0 \text{address}.\neg\text{AddressShape}) \wedge\\ &\quad\ (\leq_0 \text{friend}.\neg\text{PersonShape})\end{aligned}$$

The semantics given in the SHACL specification needs to be extended in order to handle recursive shapes. To this end, we follow the same path as in [3], and consider the evaluation of a constraint formula *given a shape assignment*, which may be intuitively viewed as labeling nodes of the graph with (sets of) shape names. Then a graph is valid iff *there exists* an assignment verifying targets and constraints. For instance, one may validate graph G_3 in Example 1, by assigning `:PersonShape` to `:p1` and `:p6`, and `:AddressShape` to `:a2`.

How to handle non-stratified constraints? For an input graph G and set S of shapes, let atoms(G, S) be the set of all atoms of the form $s(v)$ built from shape names in S and nodes in G. Then a shape assignment may be viewed as a function σ from atoms(G, S) to truth values. But should σ be required to be a *total* function to $\{\texttt{true}, \texttt{false}\}$? Such semantics may not behave well in the presence of non-stratified negation, as illustrated in Example 2. The only target node is v_1, for shape s_1. v_1 has an r_1-successor (namely v_2) in G, therefore it verifies the left disjunct $\geq_1 r_1.\top$ of the constraint for s_1. So intuitively, one would want to validate this unique target, and therefore the whole graph. But $s_2(v_1)$ cannot be assigned a truth value which complies with the constraint for s_2. If $\sigma(s_2(v_1)) = \texttt{true}$, then v_1 does not verify $\geq_1 r_2.\neg s_2$ given σ. Conversely, if $\sigma(s_2(v_1)) = \texttt{false}$, then v_1 does not violate $\geq_1 r_2.\neg s_2$ given σ.

Example 2.
$G = \{(v_1, r_1, v_2), (v_1, r_2, v_1)\}$
$s_1 \doteq\ (\geq_1 r_1.\top) \vee (\geq_1 r_2.s_2)$
$s_2 \doteq \geq_1 r_2.\neg s_2$
target: v_1 for s_1

A solution is to allow σ to be a *partial* function, or equivalently a total function from atoms(G, S) to $\{\texttt{true}, \texttt{unknown}, \texttt{false}\}$, in order to define constraint evaluation given σ. This is a key difference from the approach followed in [3]. For instance, in Example 2, the assignment $\sigma = \{s_1(v_1) \mapsto \texttt{true}, s_2(v_1) \mapsto \texttt{unknown}, s_1(v_2) \mapsto \texttt{false}, s_2(v_2) \mapsto \texttt{false}\}$ could be used to validate the graph. Such 3-valued semantics should still comply with the SHACL specification in the non-recursive case though, which may not be trivial for constraints with counting.

How to define partial validation? The SHACL specification describes a graph validation mechanism (for non-recursive shapes) which starts from a target atom, and propagates the constraints required by this target only. One way to generalize this approach to recursive shapes is to allow assigning `unknown` to an atom, even though the corresponding constraint evaluates to `true` or `false`. For instance, in Example 2,

assigning a truth value to $s_1(v_2)$ and $s_2(v_2)$ seems unnecessary for the (unique) target under validation. So could one also validate the graph with another assignment $\sigma' = \{s_1(v_1) \mapsto \texttt{true}, s_2(v_1) \mapsto \texttt{unknown}, s_1(v_2) \mapsto \texttt{unknown}, s_2(v_2) \mapsto \texttt{unknown}\}$, even though $s_1(v_2)$ and $s_2(v_2)$ evaluate to `false` given σ'? And in which cases (if any) do these two semantics differ?

Is validation scalable? Our investigations so far indicate that graph validation under such requirements is intractable in data complexity (i.e. when the constraints are assumed to be fixed), even for constraints with stratified negation, as opposed to seemingly comparable frameworks, such as stratified datalog [4]. To overcome this limitation, we are currently investigating two options: tractable fragments of the constraint language on the one hand, and approximation algorithms on the other hand.

Acknowledgements. This work was supported by the QUEST, ROBAST and OBATS projects at the Free University of Bozen-Bolzano, and the Millenium Institute for Foundational Research on Data, Chile.

References

[1] I. Boneva. Comparative expressiveness of ShEx and SHACL (Early working draft), 2016.
[2] I. Boneva, J. E. L. Gayo, S. Hym, E. G. Prud'hommeau, H. R. Solbrig, and S. Staworko. Validating RDF with shape expressions. *CoRR*, abs/1404.1270, 2014.
[3] I. Boneva, J. E. L. Gayo, and E. G. Prud'hommeaux. Semantics and Validation of Shapes Schemas for RDF. In *ISWC*, 2017.
[4] E. Dantsin, T. Eiter, G. Gottlob, and A. Voronkov. Complexity and expressive power of logic programming. *ACM Comput. Surv.*, 33(3):374–425, 2001.
[5] P. F. Patel-Schneider. Using Description Logics for RDF Constraint Checking and Closed-World Recognition. In *AAAI*, 2015.
[6] P. F. Patel-Schneider and E. Franconi. Ontology constraints in incomplete and complete data. In *ISWC*, 2012.
[7] S. Staworko, I. Boneva, J. E. Labra Gayo, S. Hym, E. G. Prud'hommeaux, and H. Solbrig. Complexity and Expressiveness of ShEx for RDF. In *ICDT*, 2015.
[8] J. Tao, E. Sirin, J. Bao, and D. L. McGuinness. Integrity Constraints in OWL. In *AAAI*, 2010.