A Tractable Notion of Stratification for SHACL

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. This article introduces a restriction on the usage of negation in SHACL "core constraint components" constraints, called strict stratification, which guarantees tractability of graph validation.

1 Introduction

One of the challenges of recent RDF-based applications is managing data quality [1], and several systems already provide RDF validation procedures (e.g., `https://www.stardog.com/docs/`, `https://www.topquadrant.com/technology/shacl/`). This created the need for a standardized declarative constraint language for RDF, and for mechanisms to detect violations of such constraints. An important step in this direction is SHACL, or Shapes Constraint Language (`https://www.w3.org/TR/shacl/`) which has become a W3C recommendation in 2017.

The SHACL specification however leaves explicitly undefined the validation of recursive constraints. In a previous article [2], we showed that extending the specification's semantics to accommodate for recursion leads to intractability (in the size of the graph) for the so-called "core constraint components" of SHACL. This result holds for stratified constraints already, which may come as a surprise, considering that stratification guarantees tractability in well-studied recursive languages such as Datalog.

Our previous work identified a tractable fragment of SHACL's core components. In this paper, we propose an alternative approach to gain tractability, retaining all SHACL operators, but strengthening the stratification condition traditionally used in logic programming. More exactly, we introduce a syntactic condition on shape constraints called "strict stratification", which guarantees that graph validation is in PTIME in combined (i.e. graph and constraints) complexity. We also describe a procedure to perform such validation.

The current paper is not self-contained, due to space limitations, but all definitions can be found in our previous article [2] or its online extended version [3].

2 SHACL Specification and Formal Semantics

We briefly recall some notions introduced in [2, 3]. We defined a graph validation problem which captures the target-based validation procedure depicted in the SHACL specification. An instance of the validation problem is a tuple $\langle G, S, s_0, v_0 \rangle$, where G is a labeled graph, $S = \{s_0 \doteq \phi_{s_0}, \ldots, s_n \doteq \phi_{s_n}\}$ is a set of shape definitions (also called "shapes" in what follows), s_0 is a shape defined in S, and v_0 is a node in G. The problem consists in deciding whether v_0 *verifies* the constraint ϕ_{s_0} for s_0, given S and G. We will call $\langle s_0, v_0 \rangle$ the *target* of this instance. Each constraint ϕ_{s_i} follows the following syntax:

$$\phi \ ::= \ \top \mid s \mid I \mid \phi_1 \wedge \phi_2 \mid \neg\phi \mid \ \geq_n r.\phi \mid \mathrm{EQ}(r_1, r_2)$$

where $\top$ means boolean true, s is a shape name defined in S, I is an IRI, r is a SPARQL property path, and $n \in \mathbb{N}^+$. Further, $\wedge$ stands for conjunction, $\neg$ for negation, "$\geq_n r.\phi$" means "must have at least n r-successors in G verifying ϕ", and "EQ(r_1, r_2)" means that the r_1 and r_2-successors of a node must coincide. A translation from SHACL core constraint components to this syntax and conversely can be found in [3].

According to this syntax, a shape can reference another. But the SHACL specification leaves open the recursive case, i.e. when a shape references itself, either directly, or via a reference cycle. As a solution, we proposed a semantics based on the notion of *assignment*, that maps (positive and negative) shape labels to sets of nodes, and we defined constraint evaluation *given an assignment*. Formally, a shape assignment for G and S is a total function from pairs $\langle s, v \rangle$ of shape name and node to $\{0(\text{false}), 0.5(\text{unknown}), 1(\text{true})\}$ (the "unknown" value is used to deal with cases when recursion and negation can be arbitrarily combined). The evaluation of formula ϕ at node v given assignment σ is written $[\![\phi]\!]^{v,G,\sigma}$ (for a formal definition we refer to [2] or [3]). The input $\langle G, S, s_0, v_0 \rangle$ is valid ff *there exists* a shape assignment σ for G and S such that *(i)* $\sigma(s_0, v_0) = 1$, and *(ii)* $\sigma(s, v) = [\![\phi_s]\!]^{v,G,\sigma}$ for each shape s defined in S and node v in G.

3 Strictly Stratified SHACL and Validation

This section defines strict stratification of a set of SHACL shapes, and proposes a PTIME validation algorithm (combined complexity) for this case. We chose to convey the general intuition behind stratification from a procedural perspective, as follows: if there is a satisfying assignment for a given input, then it may be built stratum by stratum, starting from the lowest stratum, and assigning shapes in a greedy fashion, without the need to backtrack from a higher to a lower stratum. However, adopting the classical definition of stratified negation (borrowed from Datalog) is not sufficient to avoid such backtracking. This is why we define the stronger notion of *strict* stratification.

Let S be a set of shapes. We say that shape s_1 *references* shape s_2 if s_2 appears in the definition of s_1. Further, we define the *dependency graph* of S as the directed graph with all shape names defined in S as nodes, and an edge from s_1 to s_2 iff s_1 references s_2. If the reference is in the scope of negation, then the edge is labeled with a negation symbol (as in Fig. 1), and we call it a *negative* edge. A path between two nodes is called *negative* if it contains at least one negative edge, and *positive* otherwise. S is *stratified* iff its dependency graph has no negative cycle. Finally, we define the *contracted dependency graph* of S as the graph obtained from its dependency graph by contracting all positive strongly connected components.

Definition 1 (strict stratification). *A set S of shapes is* strictly stratified *if, for any pair (s_1, s_2) of nodes in its contracted dependency graph, either:*
- *there is at most one path from s_1 to s_2, or*
- *all paths from s_1 to s_2 are positive.*

In this definition, using the contracted dependency graph (rather than the dependency graph) allows us to capture a more expressive fragment of SHACL. We also note that from this definition, a strictly stratified set of shapes must be stratified.

Example 1. Consider the three shape definitions and their contracted dependency graph in Figure 1. This set of shape is stratified, but not strictly stratified, since there are more than one paths from s_0 to s_2, and one of them is negative.

$$
\begin{aligned}
s_0 &\doteq (\geq_1 P.s_1) \wedge (\geq_1 P.s_2) \\
s_1 &\doteq \neg(\geq_1 P.s_2) \\
s_2 &\doteq (\geq_1 P.s_2)
\end{aligned}
$$

$$\begin{tikzcd}
s_1 \arrow[dr, "\neg"] & \\
s_0 \arrow[u] \arrow[r] & s_2
\end{tikzcd}$$

Fig. 1. A set of shapes, and its contracted dependency graph

$$\begin{tikzcd}
v_1 \arrow[r, "P"] & v_2 \arrow[loop above, "P"] \\
v_0 \arrow[u, "P"] \arrow[ur, "P"] \arrow[r, "P"] & v_3 \arrow[loop above, "P"]
\end{tikzcd}$$

Fig. 2. A graph under validation

Example 2. Consider again the shapes from Example 1, together with the graph in Figure 2. Assume now that one tries to build a validating assignment σ for the target $\langle v_0, s_0\rangle$ in a greedy fashion, taking advantage of stratification, thus starting from the lowest stratum $\{s_2\}$. Then s_2 can be assigned to v_0, v_2 and v_3. Moving up to the next stratum $\{s_0, s_1\}$, s_0 cannot be assigned to v_0, because the definition of s_1 is violated by v_1, and therefore the conjunct $(\geq_1 P.s_1)$ is not verified by v_0. In order to find a validating assignment, one would need to backtrack, reverting the decision to assign s_2 to v_2. But the decision to assign s_2 to v_3 on the other hand should not be reverted, otherwise the conjunct $(\geq_1 P.s_2)$ would not be verified by v_0 anymore. It can be easily seen that such pattern can lead to a blowup of backtracking alternatives, exponential in the size of the graph under validation.

The intuition for intractability is illustrated by Example 2 (in terms of backtracking), and was made more formal in [3], with a reduction from boolean circuit satisfiability. Intractability does not hold for strictly stratified shapes though, which intuitively guarantees that no backtracking is needed. We make this more precise in the following (a complete formalization with proofs can be found in [3]).

Let $\Sigma^{G,S}$ be the family of all (3-valued) assignments for G and S. The operator $\mathbf{T}^{G,S}$: $\Sigma^{G,S} \rightarrow \Sigma^{G,S}$ takes an assignment σ, and returns the assignment obtained by evaluating each shape constraint definition at each node given σ, i.e. $(\mathbf{T}^{G,S}(\sigma))(s, v) = [\![\phi_s]\!]^{v,G,\sigma}$ for each s and v. So from the definition of graph validation given in Section 2, $\langle G, S, s_0, v_0\rangle$ is valid iff $\mathbf{T}^{G,S}$ admits a fixed-point σ verifying $\sigma(s, v) = 1$. We also need the

Algorithm 1 VALIDATION PROCEDURE FOR STRICTLY STRATIFIED SHAPES

```
Input: $G, S, s_0, v_0$
1: Initiate $\sigma$ with $\sigma(s, v) = 0.5$, for each shape $s$ and node $v$
2: repeat
3:     $\sigma' \leftarrow \sigma$
4:     $\sigma \leftarrow \mathbf{T}^{G,S}(\sigma)$
5: until $\sigma = \sigma'$
6: if $\sigma(s_0, v_0) = 0$ then return Invalid
7: else return Valid
```

partial order $\preceq$ over $\Sigma^{G,S}$, defined by $\sigma_1 \preceq \sigma_2$ iff $\sigma_1(s, v) = 0$ implies $\sigma_2(s, v) = 0$, and $\sigma_1(s, v) = 1$ implies $\sigma_2(s, v) = 1$, for any s and v.

Next, it is shown in [3] that $\mathbf{T}^{G,S}$ must admit a unique minimal fixed-point σ_{minFix} w.r.t $\preceq$ over $\Sigma^{G,S}$, and that σ_{minFix} can be reached by recursive applications of $\mathbf{T}^{G,S}$, starting with the empty assignment. Finally, $\mathbf{T}^{G,S}$ is monotone w.r.t. $\preceq$, which guarantees that σ_{minFix} can be computed in polynomial time. So if $\sigma_{\text{minFix}}(s_0, v_0) = 1$, because σ_{minFix} is a fixed-point of $\mathbf{T}^{G,S}$, the graph is valid. If $\sigma_{\text{minFix}}(s_0, v_0) = 0$, because σ_{minFix} is a minimal fixed-point of $\mathbf{T}^{G,S}$, any other fixed-point σ' must verify $\sigma'(s_0, v_0) = 0$, therefore the graph is invalid. Finally, for the case $\sigma_{\text{minFix}}(s_0, v_0) = 0.5$, we showed in [3] that if S is strictly stratified (thus regardless of G), then $\Sigma^{G,S}$ must contain a fixed-point σ'' of $\mathbf{T}^{G,S}$ s.t. $\sigma''(s_0, v_0) = 1$. The corresponding validation procedure is shown in Algorithm 1.

4 Discussion and Future Work

As a possible continuation, we observe that strict stratification is only a sufficient condition for tractability. This means that the requirement may be relaxed, identifying a wider class of instances which can be validated in polynomial time. Our preliminary investigations indicate that additional properties of the dependency graph may be used, such as odd/even number of negative references in a path, or morphisms between multiple paths from one node to another.

We are also currently developing implementation techniques for validation based on logic programming. More exactly, since validation is based on the the existence of a fixed-point evaluation/assignment, we are investigating an encoding into Datalog for the strictly stratified case, and into to more expressive logic formalisms for non-strictly stratified constraints.

Acknowledgments. This work was supported by the QUEST, QUADRO and ADVANCE4KG projects at the Free University of Bozen-Bolzano.

Bibliography

[1] M. Arenas, C. Gutierrez, and J. Pérez. Foundations of RDF databases. In *Reasoning Web International Summer School*, pages 158–204. Springer, 2009.

[2] J. Corman, J. L. Reutter, and O. Savković. Semantics and Validation of Recursive SHACL. In *ISWC*, 2018.

[3] J. Corman, J. L. Reutter, and O. Savković. Semantics and Validation of Recursive SHACL. Technical Report KRDB18-1, Free University of Bozen-Bolzano, 2018. Available at `https://www.inf.unibz.it/krdb/KRDB%20files/tech-reports/KRDB18-01.pdf`.