Explanations for Non-Validation in SHACL using Answer Set Programming

Shqiponja Ahmetaj
shqiponja.ahmetaj@tuwien.ac.at

September 13, 2023

Abstract

This abstract summarizes our recent works [1, 2] on explaining the non-validation of SHACL constraints. In our framework, non-validation is explained using the notion of a *repair*, which is a collection of additions and deletions of facts that will cause the data to be consistent with the given constraints. We define a collection of decision problems for reasoning about explanations and analyze their computational complexity. We propose an algorithm to compute repairs by encoding the explanation problem – using Answer Set Programming (ASP) – into a logic program, the answer sets of which correspond to (minimal) repairs.

1 Introduction

The Shape Constraint Language (SHACL) is a recently standardized language for expressing constraints on RDF graphs. It is the result of industrial and academic efforts to provide solutions for checking the quality of RDF graphs and for declaratively describing (parts of) their structure. We recommend [10] for an introduction to SHACL and its close relative *ShEx*. The SHACL standard provides a syntax for writing down constraints, as well as describes the way RDF graphs should be *validated* w.r.t. to a given set of SHACL constraints. However, some aspects of validation were not completely specified in the standard, like the semantics of validation for constraints with cyclic dependencies. To address these shortcomings, several formalizations of SHACL based in logic-based languages with clear semantics have recently emerged [8, 3, 11].

In SHACL, the basic computational problem is to check whether a given RDF graph G *validates* a SHACL document (C, T), where C is a specification of validation rules (*constraints*) and T is a specification of nodes to which the validation rules should apply (*targets*). In order to make SHACL truly useful and widely accepted, we need automated tools that implement not only validation, which results in "yes" or "no" answers, but also support the users in their efforts to understand the reasons *why* a given graph validates or not against a

given document. The SHACL specification stresses the importance of explaining validation outcomes and introduces the notion of *validation reports* for this purpose. If a graph validates a document, the standard provides clear instructions regarding the appearance of validation reports. However, the principles of validation reports in case of non-validation are left largely open in the standard. It is not hard to see that, in general, there may be a very large number of possible reasons for a specific validation target to fail, and it is far from obvious what should be presented to the user in validation reports.

We advocate explanations in the style of *database repairs* [4] as one concrete way to provide explanations for the non-validation of SHACL constraints. This approach is closely related to abductive reasoning, model-based diagnosis, and counterfactuals, which have received significant attention in the last decades and have been applied to a range of similar problems requiring explanatory services (see, e.g., [9, 12, 6, 7]).

◦ To explain non-validation of a SHACL document (C, T) by an RDF graph G, we introduce the notion of a *SHACL Explanation Problem (SEP)*. A solution to a SEP is a pair (A, D), where A and D are sets of facts to be added and removed from G, respectively, so that the resulting graph does validate the document (C, T). We consider natural preference orders over explanations, and study also explanations that are minimal w.r.t. set inclusion and cardinality. We define a collection of inference services, which are reminiscent of basic reasoning problems in logic-based abduction [9], and study both combined and data complexity of these tasks. We then turn our attention to *non-recursive* SHACL constraints and show that, in general, reasoning does not become easier.

◦ We propose to compute repairs of a data graph by encoding the problem into ASP. We show how to transform a given data graph G and a SHACL shapes graph (C, T) into an ASP program P such that the answer sets (or, stable models) of P can be seen as a collection of plausible repairs of G w.r.t. the shapes graph (C, T). Since efficient ASP solvers exist (we use Clingo), this provides a promising way to generate data repairs in practice. The repair generation task is challenging because a given data graph might be repaired in many different ways. In fact, since fresh nodes could be introduced during the repair process, an infinite number of repairs is possible.

◦ When designing a repair generator, we need to make some choices. We have studied the scenarios, where the repair generator always introduces fresh values for cardinality constraints, and the scenario where reusing existing nodes is also allowed. We argue that forcing the generator to always introduce fresh nodes can sometimes leave out expected (minimal) repairs and even not produce any repairs at all. Indeed, there are cases when reusing existing nodes may be desired and even necessary. We have implemented and tested these encodings using the Clingo ASP system, which showed that our approach is promising for providing quality control and improvements for RDF graphs for practical use.

2 SHACL Validation

Let $\mathbf{N}$, $\mathbf{C}$, and $\mathbf{P}$ denote countably infinite, mutually disjoint sets of *nodes*, *class names*, and *property names*, respectively. A *data graph* G (RDF graph) is a finite set of *atoms* of the form $B(c)$ and $p(c,d)$, where $B \in \mathbf{C}$, $p \in \mathbf{P}$, and $c, d \in \mathbf{N}$. The set of nodes appearing in G is denoted with $V(G)$. We assume a countably infinite set $\mathbf{S}$ of *shape names*, disjoint from $\mathbf{N} \cup \mathbf{C} \cup \mathbf{P}$. A *shape atom* is an expression of the form $s(a)$, where $s \in \mathbf{S}$ and $a \in \mathbf{N}$. A *path expression* E is a regular expression built using the usual operators $*$, $\cdot$, $\cup$ from symbols in $\mathbf{P}^+ = \mathbf{P} \cup \{p^- \mid p \in \mathbf{P}\}$. If $p \in \mathbf{P}$, then p^- is the *inverse property* of p. A *(complex) shape* is an expression ϕ obeying the syntax:

$$\phi, \phi' ::= \top \mid s \mid B \mid c \mid \phi \wedge \phi' \mid \neg\phi \mid \geq_n E.\phi \mid E = E',$$

where $s \in \mathbf{S}$, $A \in \mathbf{C}$, $c \in \mathbf{N}$, n is a positive integer, and E, E' are path expressions. A *(shape) constraint* is an expression $s \leftrightarrow \phi$ where $s \in \mathbf{S}$ and ϕ is a possibly complex shape. In SHACL, *targets* are used to prescribe that certain nodes of the input data graph should validate certain shapes. W.l.o.g. we view targets as shape atoms of the form $s(a)$, where $s \in \mathbf{S}$ and $a \in \mathbf{N}$. A*shape document* is a pair (C, T), where *(i)* C is a set of constraints and *(ii)* T is a set of targets. The evaluation of a shape expression ϕ is given by assigning nodes of the data graph to (possibly multiple) shape names. A *(shape) assignment* for a data graph G is a set $I = G \cup L$, where L is a set of shape atoms such that $a \in V(G)$ for each $s(a) \in I$. The evaluation of a (complex) shape w.r.t. an assignment I is given in terms of a function that maps a (complex) shape expression ϕ to a sets of nodes, and a path expression E to a set of pairs of nodes. We refer to [**?**] for details on the evaluation of the various operators in complex shapes. For validation we consider the semantics proposed in [8]. An assignment I for G and a document (C, T) is a *(supported) model* of C if $[\![\phi]\!]^I = s^I$ for all $s \leftrightarrow \phi \in C$. The data graph G *validates* (C, T) if there exists an assignment $I = G \cup L$ for G such that (i) I is a model of C, and (ii) $T \subseteq L$.

3 Explaining Non-Validation in SHACL

In this section, we formalize the notion of explanations for non-validation of a SHACL document by a data graph, illustrate it with an example, and present some complexity results. Let G be a data graph, let (C, T) be a SHACL document, and let the set of *hypotheses* H be a data graph disjoint from G. Then $\Psi = (G, C, T, H)$ is a *SHACL Explanation Problem (SEP)*. An *explanation* for Ψ is a pair (A, D), such that (a) $D \subseteq G$, $A \subseteq H$, and (b) $(G \setminus D) \cup A$ validates (C, T).

Example 1. *Consider a SEP* $\Psi = (G, C, T, H)$*, where:*

$$\begin{aligned} C = \{&\mathsf{Teacher} \leftrightarrow \exists \mathit{teaches}.\top, \\ &\mathsf{Student} \leftrightarrow \exists \mathit{enrolledIn}.\mathit{Course} \wedge \neg\mathsf{Teacher}\} \end{aligned}$$

	$\emptyset$	$\subseteq$	$\leq$
IsExpl	NP-c	DP-c	DP-c
Exist	NP-c	NP-c	NP-c
NecAdd	coNP-c	coNP-c	$P_{\parallel}^{\mathrm{NP}}$-c
NecDel	coNP-c	coNP-c	$P_{\parallel}^{\mathrm{NP}}$-c
RelAdd	NP-c	Σ_2^P-c	$P_{\parallel}^{\mathrm{NP}}$-c
RelDel	NP-c	Σ_2^P-c	$P_{\parallel}^{\mathrm{NP}}$-c

Table 1: Complexity results for recursive SHACL constraints

$T = \{\mathsf{Student}(Ben), \mathsf{Teacher}(Ann)\}$, $H = \{Course(C_1), Course(C_2)\}$, *and* $G = \{enrolledIn(Ben, C_1), teaches(Ann, Ben), teaches(Ben, Ben), teaches(Ann, Li)\}$ *The constraints state that each* Teacher *must teach someone, and each* Student *must be enrolled in some course and must not comply with the shape* Teacher*. Note that* Teacher *and* Student *are shape names, enrolledIn is a property name, and Course is a class name. The data graph* G *validates* $(C, \{\mathsf{Teacher}(Ann)\})$*, but does not validate* (C, T)*. A possible explanation for non-validation is that* G *is missing the fact that* C_1 *is a Course; it also contains the possibly erroneous fact that* $teaches(Ben, Ben)$*. Thus, validation is ensured by repairing* G *with the explanation* (A, D)*, where* $A = \{Course(C_1)\}$ *and* $D = \{teaches(Ben, Ben)\}$*.*

We consider *preference relations* over explanations, given by a reflexive and transitive relation $\preceq$ on the set of explanations and study two typical preference orders: *subset-minimal* ($\subseteq$), and *cardinality-minimal* ($\leq$) explanations. We now define the main decision problems for SEPs. Let $\Psi = (G, C, T, H)$ be a SEP, let A, D be data graphs, let α be an atom in $G \cup H$, and let $\preceq$ be a preorder. We define six decision problems: 1) $\preceq$-IsExpl checks whether (A, D) is a $\preceq$-explanation for Ψ, 2) $\preceq$-Exist checks whether there exists a $\preceq$-explanation for Ψ 3) $\preceq$-NecAdd and 4) $\preceq$-NecDel check whether α occurs in A or D, respectively, in *every* $\preceq$-explanation (A, D) for Ψ, 5) $\preceq$-RelAdd and 6) $\preceq$-RelDel check whether α occurs in A or D, respectively, in *some* $\preceq$-explanation (A, D) for Ψ.

We present in Table 1 only the results for recursive SHACL and refer to [1] for the non-recursive fragment and for SEPs with restricted explanation signature. We omit $\preceq$ from the name of decision problems when $\preceq$ is empty, and write $(\preceq)$ when considering the variants with and without $\preceq$. We use $\preceq$ as a placeholder for both $\subseteq$ and $\leq$.

4 Encoding into ASP

To compute minimal explanations, we adapt in [2] an approach from databases and logic programming (see [5] for details) that computes minimal repairs for Datalog programs with negation. We provide here a high-level description of the encoding and refer to [2] for more details.

For a repair problem Ψ, where C is a set of non-recursive SHACL constraints in normal form, we construct a program P_Ψ, such that the stable models of (G, P_Ψ) will provide repairs for Ψ. In a nutshell, for every constraint specified by a shape in the shapes graph, the repair program P_Ψ consists of four kinds of rules: $P_{\text{Annotation}}$ consists of rules that collect existing atoms or atoms that are proposed to be in the repaired data graph, P_{Repair} consists of rules that repair the constraints by proposing additions and deletions of atoms, $P_{\text{Interpretation}}$ consists of rules that collect all the atoms that will be in the repaired data graph, and $P_{\text{Constraints}}$ consists of rules that filter out models that do not provide repairs. Intuitively, the repair program implements a top-down target-oriented approach and starts by first making true all the shape atoms in the target. From this on, the rules for constraints specified by the shapes capture violations on the targets in the rule body and propose repairs in the rule head using the annotations described above. The rules will add annotated atoms which represent additions and deletions that can be applied to the data graph to fix the violations.

We first present the basic encoding of the repair task, where the program tries to find a repair that satisfies *all* targets of the input shapes graph. This encoding employs a particular strategy for introducing new nodes in the data graph: when a value for a property needs to be added, a fresh value is always introduced. We argue that it is a reasonable strategy; it is also closely related to the standard notion of *Skolemization*. By using some of the features of ASP, we ensure that our repair program generates repairs that are minimal in terms of cardinality, which means that they contain only minimal modifications for resolving constraint violations. Our basic encoding is later extended to allow for the introduction of fresh nodes as well as the reuse of existing or previously introduced nodes.

We observe that requiring a repair to resolve violations for *all* specified targets may be too strong. In the context of the basic encoding, if the data graph has one inherently unfixable target (e.g., because of some erroneous constraint), then the repair program will have no answer sets and it will provide no guidance on how to proceed with fixing the data graph. To address this issue, we introduce the notion of *maximal repairs*, which repair the highest number of targets that is possible to repair. We show how our encoding can be augmented to generate repairs according to this new notion. This is done using the optimization features of Clingo as well as rules that allow to skip some targets.

References

[1] Shqiponja Ahmetaj, Robert David, Magdalena Ortiz, Axel Polleres, Bojken Shehu, and Mantas Simkus. Reasoning about explanations for non-validation in SHACL. In *Proceedings of KR 2021*, pages 12–21, 2021.

[2] Shqiponja Ahmetaj, Robert David, Axel Polleres, and Mantas Simkus. Repairing SHACL constraint violations using answer set programming. In *The Semantic Web - ISWC 2022*, volume 13489 of *Lecture Notes in Computer Science*, pages 375–391. Springer, 2022.

[3] Medina Andresel, Julien Corman, Magdalena Ortiz, Juan L. Reutter, Ognjen Savkovic, and Mantas Šimkus. Stable model semantics for recursive SHACL. In *Proc. of The Web Conference 2020*, WWW '20, page 1570–1580. ACM, 2020.

[4] Marcelo Arenas, Leopoldo E. Bertossi, and Jan Chomicki. Consistent query answers in inconsistent databases. In *Proc. of PODS*, pages 68–79. ACM Press, 1999.

[5] Leopoldo E. Bertossi. *Database Repairing and Consistent Query Answering*. Synthesis Lectures on Data Management. Morgan & Claypool Publishers, 2011.

[6] Diego Calvanese, Magdalena Ortiz, Mantas Simkus, and Giorgio Stefanoni. Reasoning about explanations for negative query answers in DL-Lite. *J. Artif. Intell. Res.*, 48:635–669, 2013.

[7] İsmail İlkan Ceylan, Thomas Lukasiewicz, Enrico Malizia, Cristian Molinaro, and Andrius Vaicenavicius. Explanations for negative query answers under existential rules. In *Proc. of KR 2020*, pages 223–232, 2020.

[8] Julien Corman, Juan L. Reutter, and Ognjen Savkovic. Semantics and validation of recursive SHACL. In *Proc. of ISWC'18*. Springer, 2018.

[9] Thomas Eiter and Georg Gottlob. The complexity of logic-based abduction. *J. ACM*, 42(1):3–42, 1995.

[10] José Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, and Dimitris Kontokostas. *Validating RDF Data*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers, 2017.

[11] Martin Leinberger, Philipp Seifer, Tjitze Rienstra, Ralf Lämmel, and Steffen Staab. Deciding SHACL shape containment through description logics reasoning. In *Proc. of ISWC 2020*, volume 12506 of *Lecture Notes in Computer Science*, pages 366–383. Springer, 2020.

[12] Frank Van Harmelen, Vladimir Lifschitz, and Bruce Porter. *Handbook of knowledge representation*. Elsevier, 2008.