Explanations for Non-validation in SHACL*

Shqiponja Ahmetaj[2], Robert David[2,4], Magdalena Ortiz[1], Axel Polleres[2,3], Bojken Shehu[5], and Mantas Šimkus[1]

[1] Technical University of Vienna
[2] Vienna University of Economics and Business
[3] Complexity Science Hub Vienna, Austria
[4] Semantic Web Company
[5] Polytechnic University of Tirana

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 [9] for an introduction to SHACL and its close relative *ShEx*. Among other, 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 grounded on logic-based languages with clear semantics have recently emerged [7,2,11].

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 has clear guidance how the validation reports should look like. However, the situation is different when the graph does not validate. The principles of validation reports in case of non-validation are left largely open in the standard, which specifies little beyond requiring that the node and constraint violated are indicated. 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. This is precisely the topic of our study.

* Partially supported by the Vienna Business Agency and the Austrian Science Fund (FWF) projects P30360 and P30873. Axel Polleres' work is supported by funding in the European Commission's Horizon 2020 Research Programme under Grant Agreement Number 957402.

We present here an extended abstract of [1] and additionally introduce a prototype system to automatically generate explanations. We advocate explanations in the style of *database repairs* [3] 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 very significant attention in last decades and have been applied to a range of similar problems requiring explanatory services (see, e.g., [8,12,5,6]). Our main goal is to formalize the notion of explanations for SHACL, to define a collection of reasoning tasks for exploring explanations, and to characterize their computational complexity. The main contributions of [1] are as follows:

◦ 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 or w.r.t. cardinality.

◦ We define a collection of inference services, which are reminiscent of basic reasoning problems in logic-based abduction [8], 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.

◦ Finally, we define SEPs with *restricted explanation signatures*, which allow, e.g., to specify that some classes and properties are *read-only*. We study the complexity of the problems for this generalized version of SEPs.

1 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 $s \in \mathbf{S}$, $B \in \mathbf{C}$, $c \in \mathbf{N}$, 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 [1] for details on the evaluation of the various operators in complex shapes. For validation we consider the semantics proposed in [7]. 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) $T \subseteq L$.

2 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.

Definition 1. *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 C contains the constraints $\mathsf{Teacher} \leftrightarrow \exists teaches.\top$ and $\mathsf{Student} \leftrightarrow \exists enrolledIn.Course \wedge \neg\mathsf{Teacher}$, $T = \{\mathsf{Student}(Ben), \mathsf{Teacher}(Ann)\}$, $H = \{Course(C_1), Course(C_2)\}$, and G contains $enrolledIn(Ben, C_1)$, $teaches(Ann, Ben)$, $teaches(Ben, Ben)$, $teaches(Ann, Li)$. 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. We consider two typical preference orders: *subset-minimal* ($\subseteq$), and *cardinality-minimal* ($\leq$) explanations.

Definition 2. *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 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 place holder for both $\subseteq$ and $\leq$.

Theorem 1. *The following results are true for both data and combined complexity for SHACL with recursive constraints:*

- IsExpl, $(\preceq)$-Exist, RelAdd, RelDel *are* NP*-complete,*
- $\preceq$-IsExpl *is* DP*-complete,*
- $(\subseteq)$-NecAdd *and* $(\subseteq)$-NecDel *are co*NP*-complete,*
- $\leq$-NecAdd, $\leq$-NecDel, $\leq$-RelAdd, $\leq$-RelDel *are* $P_{\|}^{\text{NP}}$*-complete,*
- $\subseteq$-RelAdd, $\subseteq$-RelDel *are* Σ_2^P*-complete.*

3 A Prototype Implementation of Explanations

To compute minimal explanations, we adapt an approach from databases and logic programming (see [4] for details) that computes minimal repairs for Datalog programs with negation. It was shown in [2] that the validation problem can be encoded as an answer-set program (ASP) which computes a model to represent validation. In case of non-validation, the idea is to add rules to advice additions and removals of facts from the input graph in a way that the repaired data graph conforms to the constraints. These suggestions come as true t_a to advice addition of a fact and as false f_a to advice deletion of a fact. We use t^* to state that an atom is or becomes true, and t^{**} to state that an atom is true in the repair. Multiple suggestions can interact and can possibly get into conflict, which is then resolved by the repair rules. In a nutshell, we rewrite a SEP as an ASP program, whose models will identify minimal repairs of the input SEP. We illustrate the idea with an example.

Example 2. Consider a SEP (G, C, T, H), where $G = \{enrollIn(Ben, C_1)\}$, $C = \{\mathsf{Student} \leftrightarrow \exists enrollIn.Course\}$, $T = \{\mathsf{Student}(Ben)\}$, and $H = \{Course(C_1)\}$. The repair program consists of the following rules and facts.

First, it contains $enrollIn(Ben, Y)$ from G, and also the fact $\mathsf{Student}(Ben, t^*)$. Second, it contains rules of the form $\mathsf{enrollIn\_}(X, Y, t^*) \leftarrow \mathsf{enrollIn\_}(X, Y, t_a) \vee \mathsf{enrollIn}(X, Y)$ that capture the intended meaning of annotations (similarly also for *Course*). Roughly, they say that if an atom is true in the data or is advised to be true, then it should be annotated with t^*. Note that we use fresh predicates $\mathsf{enrollIn\_}$, and $\mathsf{Course\_}$, which extend the arity of the original ones with 1. Third, it contains the *repair rules*: $\mathsf{enrollIn\_}(X, Y, t_a) \leftarrow \mathsf{Student\_}(X, t^*) \wedge \neg\mathsf{enrollIn}(X, Y)$, and $\mathsf{Course\_}(Y, t_a) \leftarrow \mathsf{Student\_}(X, t^*) \wedge \mathsf{enrollIn\_}(X, Y, t^*) \wedge \neg\mathsf{Course}(X)$. The bodies of these rules capture the possible ways to invalidate the target atom, and the heads capture the possible ways to restore validation by advising to add or remove atoms from the data. For simplicity, we discard in this example the irrelevant atoms with f_a. Finally, we have the interpretation rules of the form $\mathsf{enrollIn\_}(X, Y, t^{**}) \leftarrow \mathsf{enrollIn\_}(X, Y, t^*)$ (similarly for $\mathsf{Course\_}$), which add at the end the atoms that should be true in the repair. The model of this program contains $Course\_(C_1, t_a)$, $Course\_(C_1, t^*)$, $Course\_(C_1, t^{**})$, $enrolledIn\_(Ben, C_1, t^*)$, and $enrolledIn\_(Ben, C_1, t^{**})$. The annotations state that both $enrollIn(Ben, C_1)$ and $Course(C_1)$ should be in the repaired data graph.

Implementation details The implementation is done by a Java program that produces logic programs for Clingo based on an approach described in [10]. In a nutshell, the process is done in three steps: 1) We convert RDF triples and SHACL constraints into an ASP program that runs the SHACL validation and produces the validation report as minimal models of the program. 2) In case of non-validation, we add the annotated repair rules described above. Thus, the program is extended to produce the repairs in addition to the validation report. 3) The extended logic program is then solved using Clingo and produces the repair proposals from the stable models of the program. These repairs minimally change the input data graph through additions and deletions of facts.

References

1. Ahmetaj, S., David, R., Ortiz, M., Polleres, A., Shehu, B., Šimkus, M.: Reasoning about explanations for non-validation in SHACL. In: Proceedings of the 18th International Conference on Principles of Knowledge Representation and Reasoning, KR 2021. To appear.
2. Andresel, M., Corman, J., Ortiz, M., Reutter, J.L., Savkovic, O., Šimkus, M.: Stable model semantics for recursive SHACL. In: Proc. of The Web Conference 2020. p. 1570–1580. WWW '20, ACM (2020). https://doi.org/10.1145/3366423.3380229
3. Arenas, M., Bertossi, L.E., Chomicki, J.: Consistent query answers in inconsistent databases. In: Proc. of PODS. pp. 68–79. ACM Press (1999). https://doi.org/10.1145/303976.303983
4. Bertossi, L.E.: Database Repairing and Consistent Query Answering. Synthesis Lectures on Data Management, Morgan & Claypool Publishers (2011). https://doi.org/10.2200/S00379ED1V01Y201108DTM020
5. Calvanese, D., Ortiz, M., Simkus, M., Stefanoni, G.: Reasoning about explanations for negative query answers in DL-Lite. J. Artif. Intell. Res. **48**, 635–669 (2013). https://doi.org/10.1613/jair.3870
6. Ceylan, İ.İ., Lukasiewicz, T., Malizia, E., Molinaro, C., Vaicenavicius, A.: Explanations for negative query answers under existential rules. In: Proc. of KR 2020. pp. 223–232 (2020). https://doi.org/10.24963/kr.2020/23
7. Corman, J., Reutter, J.L., Savkovic, O.: Semantics and validation of recursive SHACL. In: Proc. of ISWC'18. Springer (2018). https://doi.org/10.1007/978-3-030-00671-6_19
8. Eiter, T., Gottlob, G.: The complexity of logic-based abduction. J. ACM **42**(1), 3–42 (1995). https://doi.org/10.1145/200836.200838
9. Gayo, J.E.L., Prud'hommeaux, E., Boneva, I., Kontokostas, D.: Validating RDF Data. Synthesis Lectures on the Semantic Web: Theory and Technology (2017). https://doi.org/10.2200/S00786ED1V01Y201707WBE016
10. Labra-Gayo, J.E., García-González, H., Fernández-Alvarez, D., Prud'hommeaux, E.: Challenges in rdf validation. In: Current Trends in Semantic Web Technologies: Theory and Practice, pp. 121–151. Springer (2019)
11. Leinberger, M., Seifer, P., Rienstra, T., Lämmel, R., Staab, S.: Deciding SHACL shape containment through description logics reasoning. In: Proc. of ISWC 2020. Lecture Notes in Computer Science, vol. 12506, pp. 366–383. Springer (2020). https://doi.org/10.1007/978-3-030-62419-4_21
12. Van Harmelen, F., Lifschitz, V., Porter, B.: Handbook of knowledge representation. Elsevier (2008)