Magic Shapes for Validation in SHACL[⋆]

Shqiponja Ahmetaj[2], Bianca Löhnert[1], Magdalena Ortiz[1], and Mantas Šimkus[1]

[1]TU Wien, Austria, [2]WU Wien, Austria

(POSTER)

The Shape Constraint Language (SHACL) was recently standardized by the W3C as a formalism for checking the quality of RDF graphs; we refer to [7] for an introduction. In SHACL, the main problem is to check whether a given RDF graph G *validates* a SHACL document (C, T), where C is a set of *constraints*, also called *shapes graph*, each associated to a so-called *shape name*, and T (*targets*) is a specification of nodes from the data graph which should validate certain shapes from C. For illustration, consider a graph $G = \{\mathit{enrolledIn}(\mathit{Ben}, C_1)\}$ and a SHACL document (C, T), where $C = \{\mathsf{Student} \leftrightarrow \exists \mathit{enrolledIn}.\mathit{Course}\}$, and T is the shape atom $\mathsf{Student}(\mathit{Ben})$. The constraint states that each $\mathsf{Student}$ must be enrolled in some course; $\mathsf{Student}$ is a shape name, and *enrolledIn* and *Course* are data predicates, i.e., property and class name, respectively. Clearly, G does not validate (C, T), but the extended graph $G' = G \cup \{\mathit{Course}(C_1)\}$ does.

The standard specifies a syntax for expressing SHACL constraints and describes when they are *validated* by RDF graphs. However, it leaves undefined the semantics of *recursive* constraints, i.e., constraints that involve cyclic dependencies. To address this, some logic-based proposals to formalize the semantics of full SHACL have emerged recently. Andresel et al. [1] proposed a semantics based on the stable models semantics for logic programs, stricter than the semantics based on classical logic due to Corman el al. [4]. Both semantics coincide with the official recommendation for non-recursive constraints, and unfortunately, the validation problem is NP-complete under both.

To make SHACL truly useful and facilitate its adoption, we need automated tools that efficiently implement validation and scale well in the presence of large RDF graphs and sets of constraints. There are already significant efforts in this direction for fragments of SHACL [3, 6]. Shacl2Sparql [3] is a SHACL validation engine that checks conformance of RDF graphs with SHACL constraints by evaluating SPARQL queries against the data, which optimzes the order in which shapes are processed. Further optimization techniques are implemented in Trav-Shacl [6]. However, these works focus on tractable fragments of SHACL. They do not handle *unrestricted* interaction of recursion and negation in SHACL constraints, which calls for verifying whether there exists some *global* assignment of shapes to nodes in the graph that is *consistent* with all the constraints. This cannot be easily done using top-down approaches implemented in existing validators. To our knowledge, the only implementation that validates full SHACL

⋆ Supported by the Vienna Business Agency and the Austrian Science Fund (FWF) projects P30360 and P30873, and the Faculty of Informatics, TU Wien.

is the Shacl-Asp prototype[1] from [1], which translates SHACL constraints into ASP programs and evaluates them using the DLV system[2]. We note that the authors of [3] propose an algorithm for full SHACL that involves a SAT solver, but to our knowledge, with no available implementation. The focus of this work is SHACL validation in the presence of unrestricted negation and recursion.

In this extended abstract we present preliminary work on adapting the Magic Sets technique, known from logic programs with (unstratified) negation [5], to SHACL constraints, as a way to improve performance and applicability of SHACL validators. Magic Sets [2] is a well-known optimization method from logic programming and deductive databases. In a nutshell, it uses the query goal to adorn the input program with binding information, obtaining a new program whose bottom-up evaluation, analogously to the top-down one, involves ground atoms only from the relevant part of the data. When applied to SHACL shape graphs, it allows us to do validation on a potentially much smaller fragment of the input RDF graph, while also ingoring shape constraints in the input that do not affect validation of the targets. This is particularly useful in the presence of negation and recursion, which may result in inconsistency and non-validation, even when they are irrelevant to the target of interest. We report some preliminary experiments showing that the technique significantly improves the performance of the Shacl-Asp validator, and it may also allow the exploitation of the more optimized engines Shacl2Sparql and Trav-Shacl for input shape graphs that they could not originally handle.

1 Preliminaries

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 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: $\phi, \phi' ::= \top \mid s \mid A \mid a \mid \phi \wedge \phi' \mid \neg\phi \mid \geq_n E.\phi \mid E = E'$, 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 $s \leftarrow \phi$ where $s \in \mathbf{S}$ and ϕ is a possibly complex shape; we may refer to ϕ as the body of the constraint and s as head. 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 *SHACL 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 shape names. A *(shape) assignment* for 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 shape w.r.t. I is given in terms of a function that maps a

[1] https://github.com/medinaandresel/shacl-asp
[2] http://www.dlvsystem.com/dlv/

shape expression to a set of nodes, and a path expression to a set of pairs of nodes. We refer to [1] for more details. We focus on the stable model semantics.

2 Magic Shape Algorithm for SHACL

We now briefly describe the Magic algorithm for SHACL, which takes as input a document (C, T) and outputs a new optimized shapes graph against which data graphs can be validated, see Figure 1. The method comprises four main steps:

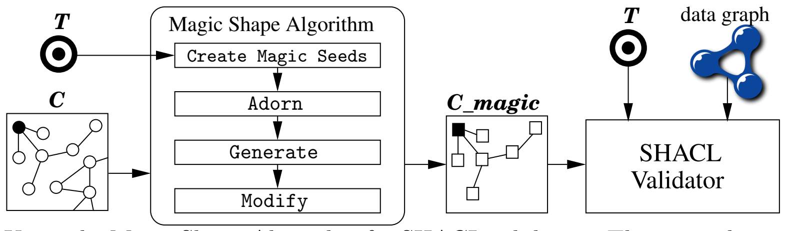

Fig. 1: Using the Magic Shape Algorithm for SHACL validation. The input shapes graph **C** and targets **T** are transformed into a (typically smaller and simpler) shapes graph **C_magic** relevant for the targets, which can be validated with off-the-shelf validators

1. `Create Magic Seeds`: for each target shape atom $s(a) \in T$, the algorithm adds a constraint of the form $s\_magic \leftarrow a$, called a *magic seed*.
2. `Adorn`: for each shape name s occurring in the targets T, an adorned shape s^b is pushed onto a stack. Procedure `Adorn` then pops s^b and propagates the adornment to the body of the constraints with head s, pushing each freshly adorned shape onto the stack.
3. `Generate`. The adorned set of constraints is then used to generate "magic" constraints through the procedure `Generate`. Roughly, for a constraint r of the form $s \leftarrow \phi$, it creates a new constraint with s_magic in the body and s'_magic in the head for each adorned shape name s' in ϕ. In case that in the constraint two shapes are connected by a path expression E, the path expression has to be inverted.
4. `Modify` enhances the body of each adorned constraint with a magic version of the shape occurring in the head of the constraint, i.e. r will be rewritten as $s \leftarrow s\_magic \wedge \phi$ and added to the set of modified constrains.

In the presence of arbitrary negation and recursion, we also need to identify the so-called *dangerous* constraints that appear under the scope of negation. If they are relevant for validating the target, their adornments are propagated also from the body to the head [5].

Example 1. Consider a SHACL document (C, T), where C consists of the constraints $s_1 \leftarrow s_2$, $s_2 \leftarrow s_3 \wedge s_4$, $s_3 \leftarrow \geq_5 p.s_5$, $s_6 \leftarrow \neg s_6$, and $T = \{s_1(a)\}$. The algorithm first adds $s_{1\_magic} \leftarrow a$. Since s_1 appears in T, the shape names s_1 to s_5 and the corresponding constraints will be adorned. The last (disconnected) constraint will not participate in the resulting program. `Generate(`r_a`)` adds $s_{2\_magic} \leftarrow s_{1\_magic}$, $s_{3\_magic} \leftarrow s_{2\_magic}$, $s_{4\_magic} \leftarrow s_{2\_magic}$ and $s_{5\_magic} \leftarrow \geq_1 p^{-}.s_{3\_magic}$. Finally, the modified constraints are: $s_1 \leftarrow s_{1\_magic} \wedge s_2$, $s_2 \leftarrow s_{2\_magic} \wedge s_3 \wedge s_4$, and $s_3 \leftarrow s_{3\_magic} \wedge \geq_5 p.s_5$.

3 Implementation and Experiments

We implemented a prototype of the Magic Shapes Algorithm. It receives as input a shapes graph (in Turtle syntax) and produces as output a new shapes graph. To evaluate the usefulness of the approach, we perform experiments with the Shacl-Asp validator. For each test case, we run the validator once with the original shapes graph and once with the magic variant.
Data Graph. The data for the experiments was obtained from DBPedia (version 2016-10)[3]. More precisely, we used the datasets “PersonData”, “Instance Types”, “Labels”, “Mappingbased Literals” and “Mappingbased Objects”. The datasets are in Turtle syntax and contain about 61 million triples (7.7 GB).
Shapes Graph. We created six shapes graphs[4] S_1 to S_6 from the domain of movies and actors, similar to those in [3]. Some of the shapes graphs are inconsistent, i.e., there is no valid shape assignment. As target we use either the class “dbo:Person” for “MusicianShape” (S_1, S_2, S_4) or the class “dbo:Film” for “MovieShape” (S_3, S_5, S_6).
Setting. The experiments were performed on a Linux server with a 24 core Intel Xeon CPU running at 2.20 GHz and 264 GB of RAM. We used the DLV based Shacl-Asp validator. An Apache Jena TDB[5] was used as an RDF triple store to retrieve the relevant triples from the data graph, which are transformed into ASP facts; this was done for the input shapes graph and for its magic version.
Results. Table 1 summarizes the results of our experiments. Although all inputs except S_1 are recursive, the output of the Magic Shapes algorithm is only recursive for S_3, S_5 and S_6. The shapes graphs S_4 to S_6 are inconsistent in general. S_4 and S_5 become consistent after running the algorithm since the constraints causing inconsistency are not relevant for validating the targets; this is not the case for S_6. In some cases the output of the algorithm falls in a fragment that is supported by Shacl2Sparql or Trav-Shacl although the original shapes graphs are not, e.g., S_4 and S_5. We distinguish the non-recursive fragment $\mathcal{L}_{non\text{-}rec}$ and the fragment $\mathcal{L}_s$, which restricts the interaction between recursion and negation. The last column shows that the Magic Shapes algorithm significantly improves the performance of the DLV validator.

4 Conclusion

In this extended abstract, we have presented initial results on adapting the Magic Set technique for improving the performance of SHACL validators. This is par-

[3] http://downloads.dbpedia.org/wiki-archive/downloads-2016-10.html
[4] https://github.com/biziloehnert/magicSHACL/tree/master/experiments
[5] https://jena.apache.org/documentation/tdb/

case	#shapes		recursive		inconsistent		$\mathcal{L}_{non\text{-}rec}$		$\mathcal{L}_s$		SHACL-ASP	
	orig.	*magic*	*orig.*	*magic*	*orig.*	*magic*	*orig.*	*magic*	*orig.*	*magic*	*orig.*	*magic*
S_1	15	6					✓	✓	✓	✓	1798s	934s
S_2	15	6	✓					✓	✓	✓	1927s	797s
S_3	15	6	✓	✓					✓	✓	1872s	582s
S_4	15	6	✓		✓			✓		✓	1965s	820s
S_5	15	6	✓	✓	✓					✓	2036s	540s
S_6	15	6	✓	✓	✓	✓					1932s	592s

Table 1: Experiment Results

ticularly useful when the input shapes graphs contain unrestricted interaction of recursion and negation. We performed experiments with Shacl-Asp, the only SHACL validator that can support such shape graphs, and showed that the simpler shapes graphs produced by the Magic Shapes algorithm are evaluated significantly more efficiently than the original ones. The algorithm may also discard constraints with recursion or negation that are not relevant for validating the targets, possibly eliminating some irrelevant inconsistency. The resulting shapes graphs may fall into SHACL fragments that can be handled by more optimized SHACL adopters such as Shacl2Sparql and Trav-Shacl, thus enabling the use of these engines on graphs that they could not originally support.

We are running some experiments to identify cases where the Magic Shapes technique can also improve the performance of Shacl2Sparql or Trav-Shacl on inputs that they can handle.

References

1. 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. 15701580. ACM (2020). https://doi.org/10.1145/3366423.3380229
2. Bancilhon, F., Maier, D., Sagiv, Y., Ullman, J.: Magic sets and other strange ways to implement logic programs (extended abstract). In: PODS '86 (1985)
3. Corman, J., Florenzano, F., Reutter, J.L., Savkovic, O.: Validating shacl constraints over a sparql endpoint. In: ISWC. Springer (2019). https://doi.org/10.1007/978-3-030-30793-6_9
4. 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
5. Faber, W., Greco, G., Leone, N.: Magic sets and their application to data integration. J. Comput. Syst. Sci. pp. 584–609 (2007). https://doi.org/10.1016/j.jcss.2006.10.012
6. Figuera, M., Rohde, P.D., Vidal, M.: Trav-shacl: Efficiently validating networks of SHACL constraints. In: WWW. pp. 3337–3348. ACM / IW3C2 (2021). https://doi.org/10.1145/3442381.3449877
7. Gayo, J.E.L., Prud'hommeaux, E., Boneva, I., Kontokostas, D.: Validating RDF Data. Synthesis Lectures on the Semantic Web: Theory and Technology, Morgan & Claypool Publishers (2017). https://doi.org/10.2200/S00786ED1V01Y201707WBE016