Magic Shapes for SHACL Validation

Shqiponja Ahmetaj
TU Wien (Austria), WU Wien (Austria)
shqiponja.ahmetaj@tuwien.ac.at

Bianca Löhnert
TU Wien (Austria)
bianca@loehnert-net.at

Magdalena Ortiz
TU Wien (Austria)
ortiz@kr.tuwien.ac.at

Mantas Šimkus
Umeå University (Sweden), TU Wien (Austria)
simkus@cs.umu.se

ABSTRACT

A key prerequisite for the successful adoption of the Shapes Constraint Language (SHACL)—the W3C standardized constraint language for RDF graphs—is the availability of automated tools that efficiently validate targeted constraints (known as *shapes graphs*) over possibly very large RDF graphs. There are already significant efforts to produce optimized engines for SHACL validation, but they focus on restricted fragments of SHACL. For *unrestricted* SHACL, that is SHACL with unrestricted recursion and negation, there is no validator beyond a proof-of-concept prototype, and existing techniques are inherently incompatible with the *goal-driven* approaches being pursued by existing validators. Instead they require a *global* computation on the entire data graph that is not only computationally very costly, but also brittle, and can easily result in validation failures due to conflicts that are irrelevant to the validation targets.

To address these challenges, we present a 'magic' transformation—based on Magic Sets as known from Logic Programming—that transforms a SHACL shapes graph S into a new shapes graph S' whose validation considers only the relevant neighbourhood of the targeted nodes. The new S' is equivalent to S whenever there are no conflicts between the constraints and the data, and in case the validation of S fails due to conflicts that are irrelevant to the target, S' may still admit a lazy, target-oriented validation. We implement the algorithm and run preliminary experiments, showing our approach can be a stepping stone towards validators for full SHACL, and that it can significantly improve the performance of the only prototype validator that currently supports full recursion and negation.

PVLDB Reference Format:
Shqiponja Ahmetaj, Bianca Löhnert, Magdalena Ortiz, and Mantas Šimkus. Magic Shapes for SHACL Validation. PVLDB, 15(10): 2284-2296, 2022.
doi:10.14778/3547305.3547329

PVLDB Artifact Availability:
The source code, data, and/or other artifacts have been made available at https://github.com/biziloehnert/magicSHACL.

1 INTRODUCTION

The Shapes Constraint Language (SHACL) was recently standardized by the W3C as a formalism for checking the quality of RDF graphs; we refer to [15] for an introduction. In SHACL, the main problem is to check whether a given RDF graph G *validates* a SHACL shapes graph (C, T), where C is a set of *constraints*, each associated to a so-called *shape name*, and T is a specification of nodes (*targets*) from the data graph which should validate certain shapes from C. For illustration, consider a SHACL shapes graph (C, T), where $C = \{\mathsf{Queen} \leftarrow \exists married.\top \wedge \exists has.crown\}$ contains one constraint stating that each Queen must be married and have a crown, and in T we have the *shape atom* $\mathsf{Queen}(Sissi)$. Here Queen is a shape name, *married* and *has* are data predicates, i.e., properties, and *crown* is a constant. The graph $G = \{married(Sissi, Franz)\}$ does not validate (C, T), but the extended graph $G' = G \cup \{has(Sissi, crown)\}$ does.

The standard specifies a syntax for expressing SHACL constraints and describes when they are *validated* by RDF graphs. It allows for *recursive* constraints, that is, constraints that involve cyclic dependencies, but their semantics is not defined in the specification and left open to implementers. This has motivated some recent logic-based proposals to formalize the semantics of full SHACL [4, 8].

All formalizations of SHACL validation—with and without recursion—are based on *assignments* of shapes to nodes in the data graph, which have to comply with the constraints and contain the targets. Such assignments are *global* in the sense that they cover all nodes in the data graph. For tractable fragments that restrict the interaction of recursion and negation, it has been shown that validation under global (a.k.a. strict faithful) assignments coincides with validation under a weaker form of *faithful assignments* that allow to decide only some shape at some nodes, and to leave others undefined. Moreover, such faithful assignments can be obtained starting from the nodes mentioned in the targets and visiting only their relevant neighbourhood, while deciding the value of assignments only for the shape names occurring in the constraints.

Unfortunately, the picture is significantly different in the presence of negation and recursion, whose interaction may hinder the existence of a valid shape assignment and thus make a data graph inconsistent with a set of SHACL constraints. Consider the following example shapes graph (C, T) and data graph G:

$$C = \{\mathsf{Crowned} \leftarrow \exists crownedBy.\neg\mathsf{Crowned}\}$$
$$T = \{\mathsf{Crowned}(Sissi)\}$$
$$G = \{crownedBy(Sissi, Archbishop), crownedBy(Tim, Tim)\}$$

$\mathsf{Crowned}(Sissi)$ satisfies the constraint for Crowned since *Sissi* is crowned by the *Archbishop*, who himself is not Crowned. Although one may expect the target to be valid, no valid global assignment exists, since there is no consistent way to assign or not assign node *Tim* to the shape name Crowned. This may not be desired

Proceedings of the VLDB Endowment, Vol. 15, No. 10 ISSN 2150-8097.
doi:10.14778/3547305.3547329

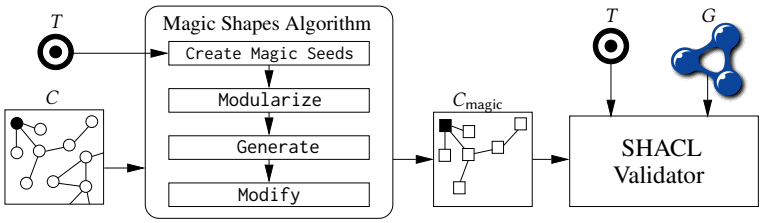


Figure 1: Using Magic Shapes for SHACL validation. The input set C and targets T are transformed into a (typically smaller and simpler) shapes graph C_{magic} relevant for the targets, which can be validated with off-the-shelf validators

since the part of the data graph relevant to validating the target Crowned(*Sissi*) is independent from *Tim* being crowned or not, and indeed, a faithful assignment could leave Crowned(*Tim*) undefined. This is a toy example, but in real life RDF graphs may be huge and errors may be unavoidable. Simply discarding the entire graph because of a problematic fact unrelated to our targets may be highly undesirable. Moreover, finding a global assignment over the full graph is costly. It makes the worst-case complexity of validation to go up from P to NP-hard [8], and additionally, it is incompatible with the target-guided top-down approaches. Indeed, recent and ongoing efforts to implement scalable SHACL validators use this type of goal-oriented approaches [7, 14], but focus on tractable fragments of SHACL, and do not handle *unrestricted* SHACL, that is with unrestricted recursion and negation. Existing algorithms search for global assignment, and only one of them is implemented in a proof-of-concept prototype [4]. To our knowledge, no target-oriented way to compute faithful assignments looking only at the relevant part of the graph has been proposed.

The focus of this work is SHACL validation in the presence of unrestricted negation and recursion. We build on *Magic Sets* [2, 3, 6], a well-known technique developed in the context of deductive databases to combine the advantages of top-down and bottom-up evaluation. 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 only the part of the data relevant to answering the query. We adapt this technique to SHACL shape graphs. Our main contribution is a *Magic Shapes* technique that takes as input a shapes graph $S = (C, T)$, and produces a new, potentially significantly smaller 'magic shapes graph' $S_m = (C_{magic}, T)$. On the one hand, C_{magic} discards the constraints in C that are not necessary for validating the target T. At the same time, it generates a set of 'magic constraints' that, in a nutshell, ensure that validation is always restricted to a relevant fragment of the input data graph, starting from the nodes directly mentioned in T, and iteratively identifying their relevant neighborhood. The output S_m can then be passed to existing validators instead of the input S, see Figure 1. Whenever the data graph is consistent with the constraints and a global assignment exists, S_m and S are equivalent. Otherwise, if there is no global assignment, the data graph may still validate the 'magic shapes graph' if the problematic part of the data graph and set of constraints is not relevant to the current target. In this way, we can find faithful assignments that leave irrelevant shape adornments undefined, enabling a more fine-grained validation than under current semantics. For tractable fragments, it also provides an easy way to do validation on a potentially much smaller fragment of the input RDF graph, while ignoring constraints that do not affect validation of the targets, independently of the algorithms implemented by the specific validator at hand.

Structure of the paper. In Section 2, we present the syntax of SHACL and the notion of validation under both the classical (or supported) and stable model semantics. We describe the Magic Shapes algorithm in Section 3 and Section 4. To illustrate the general ideas of the algorithm, we first concentrate in Section 3 on the *positive* fragment of SHACL. The full algorithm for SHACL constraints with arbitrary negation and recursion is described in Section 4. Section 5 is dedicated to showing the correctness of the Magic Shapes algorithm. We then discuss in Section 6 how the Magic Shapes technique allows us to find faithful assignments and define an inconsistency-tolerant version of the semantics. We implemented the Magic Shapes technique and did some preliminary experiments; the results are presented in Section 7.

Related Work. Logic-based proposals to formalize the semantics of full SHACL—with recursive constraints—have emerged recently. Andresel et al. [4] 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. [8]. Both semantics coincide with the official recommendation for non-recursive constraints. There is a close connection between SHACL and the more established family of *Description Logics* [5], which has been used to gain new insights into SHACL (see, e.g., [16, 20]). There has also been increasing interest in scalable SHACL validation. Shacl2Sparql [7] is a validation engine that checks conformance of RDF graphs with SHACL constraints by evaluating SPARQL queries against the data, which optimizes the order in which shapes are processed. Further optimization techniques are implemented in Trav-Shacl [14]. However, these works focus on tractable fragments of SHACL, and do not handle *unrestricted* interaction of recursion and negation in constraints. To our knowledge, the only implementation that validates unrestricted SHACL is the Shacl-Asp prototype from [4], which translates SHACL constraints into answer set programs (ASP) [12] and evaluates them using the DLV system [1]. Note that [7] proposes an algorithm for unrestricted SHACL that involves a SAT solver, but to our knowledge, with no available implementation.

Our Magic Shapes technique is inspired by the well-known Magic Set transformation for optimizing evaluation of Datalog programs in bottom-up systems [6, 18], and its extensions for Datalog with disjunction [3, 9] and (possibly stratified) negation [2, 6, 13, 21]. In paticular, we build on the ideas of the Magic Set method presented in [13] for Datalog with unstratified negation, which goes beyond optimization, and also addresses the challenge of meaningful semanics in the face of possible inconsistencies.

2 PRELIMINARIES

We present here SHACL and the notion of *validation* by RDF graphs under the supported model semantics and the stable model semantics. We follow the formalization from [4].

Data graph Let **N** and **P** denote infinite, disjoint sets of *nodes* and *property names*, respectively. A *(data) graph* G is a finite set of atoms of the form $p(a, b)$, where $p \in \mathbf{P}$ and $a, b \in \mathbf{N}$. The set of nodes appearing in G is denoted with $V(G)$.

Syntax of SHACL Let **S** be an infinite set of *shape names*, disjoint from **N** and **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 build from the operators $*$, $\cdot$, $\cup$, and symbols p or expressions p^-, where $p \in \mathbf{P}$. A *(shape) expression* is an expression ϕ of the form:

$$\phi := \top \mid s \mid a \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \geq_n E.\phi \mid E = E',$$

where $s \in \mathbf{S}$, $a \in \mathbf{N}$, $n \in \mathbb{N}$, and E, E' are path expressions. We use the abbreviations $\exists E.\phi$ for $\geq_1 E.\phi$, and $\leq_n E.\phi$ for $\neg(\geq_{n+1} E.\phi)$. A *(shape) constraint* is of the form $s \leftarrow \phi$, where $s \in \mathbf{S}$ and ϕ is a shape expression. We may sometimes call s the *head* and ϕ the *body* of the constraint. *A target set* (or simply *target*) is a set of shape atoms of the form $s(a)$ where $s \in \mathbf{S}$ and $a \in \mathbf{N}$. A *shapes graph* is a pair (C, T), where C is a set of constraints and T is a target. The *definition* $\phi_{s,C}$ of a shape name s in a set of constraints C is the disjunction of all shape expressions in the body of a shape constraint in C whose head shape is s. That is, $\phi_{s,C} = \bigvee_{s \leftarrow \phi \in C} \phi$. If C is clear from the context, we may simply write ϕ_s instead of $\phi_{s,C}$. For the theoretical part of this work, we only consider ground targets given as shape atoms, since the other types of targets (like the so-called *class targets*) can be converted to ground targets using SPARQL queries. The SHACL recommendation[1] provides a possible definition in SPARQL for each target type.

Evaluation of shape expressions An *assignment* A for a graph G is a set of shape atoms such that $v \in V(G)$ for each $s(v) \in A$. The set $G \cup A$ is called *decorated graph* for G. The evaluation of a (complex) shape expression w.r.t. a decorated graph I is given in Table 1 in terms of a function $[\![\cdot]\!]^I$ that maps a (complex) shape expression ϕ to a set of nodes, and a path expression E to a set of pairs of nodes. Throughout the paper, when validating a graph G against a shapes graph S, we assume that all constants that occur in S also appear in G.

Supported and stable model semantics We first present the supported model semantics introduced in [8].

Definition 2.1. Given a set of shape constraints C, a decorated graph I is called a *supported model* of C, if $[\![\phi_{s,C}]\!]^I = [\![s]\!]^I$ holds for each shape name s in C. A graph G *validates* a shapes graph (C, T) under *the supported model semantics* if there exists an assignment A for G such that (i) $G \cup A$ is a supported model, and (ii) $T \subseteq A$.

The stable model semantics restricts the supported model semantics by requiring that every shape atom of a decorated graph has a (non-circular) justification. This is achieved by requiring the existence of a level assignment [4].

[1] https://www.w3.org/TR/shacl/#targets

Table 1: Evaluation of shape expressions

$$[\![\top]\!]^I = V(I) \qquad [\![c]\!]^I = \{c\}$$

$$[\![p]\!]^I = \{(v, v') \mid p(v, v') \in I\} \qquad [\![p^-]\!]^I = \{(v, v') \mid p(v', v) \in I\}$$

$$[\![E \cup E']\!]^I = [\![E]\!]^I \cup [\![E']\!]^I \qquad [\![E \cdot E']\!]^I = [\![E]\!]^I \circ [\![E']\!]^I$$

$$[\![E^*]\!]^I = \{(v, v) \mid v \in V(I)\} \cup [\![E]\!]^I \cup [\![E \cdot E]\!]^I \cup [\![E \cdot E \cdot E]\!]^I \cup \cdots$$

$$[\![s]\!]^I = \{v \mid s(v) \in I\} \qquad [\![\neg\phi]\!]^I = V(I) \setminus [\![\phi]\!]^I$$

$$[\![\phi_1 \vee \phi_2]\!]^I = [\![\phi_1]\!]^I \cup [\![\phi_2]\!]^I \qquad [\![\phi_1 \wedge \phi_2]\!]^I = [\![\phi_1]\!]^I \cap [\![\phi_2]\!]^I$$

$$[\![\geq_n E.\phi]\!]^I = \{v \mid |\{(v, v') \in [\![E]\!]^I \text{ and } v' \in [\![\phi]\!]^I\}| \geq n\}$$

$$[\![E = E']\!]^I = \{v \mid \forall v' : (v, v') \in [\![E]\!]^I \text{ iff } (v, v') \in [\![E']\!]^I\}$$

Definition 2.2 (Level assignment). Let I be supported model. A *level assignment* for I is a function level that maps tuples in $\{(\phi, v) \mid v \in [\![\phi]\!]^I\}$ to integers, and satisfies the following conditions:

(i) $level(\phi_1 \wedge \phi_2, v) = max(\{level(\phi_1, v), level(\phi_2, v)\})$
(ii) $level(\phi_1 \vee \phi_2, v) = min(\{level(\phi_1, v), level(\phi_2, v)\})$
(iii) $level(\geq_n E.\phi, v)$ is the smallest $k \geq 0$ for which there exist n nodes $v_1, v_2, \ldots, v_n$ such that for all $1 \leq i \leq n$
  (a) $(v, v_i) \in [\![E]\!]^I$, $v_i \in [\![\phi]\!]^I$, and
  (b) $level(\phi, v_i) \leq k$.

Definition 2.3 (Stable Model Semantics). A decorated graph I is a stable model of a set C of constraints, if (i) I is a supported model of C, and (ii) there exists a level assignment such that for all $s(v) \in I$ $level(\phi_s, v) < level(s, v)$. A set of shape constraints C is called *consistent* with G under the supported (stable) model semantics if there exists a supported (stable) model of C; otherwise C is called *inconsistent* with G (under the supported (stable) model semantics).

A data graph G *validates* a shapes graph (C, T) under the stable model semantics if there *exists* an assignment A such that (i) $G \cup A$ is a stable model of C, and (ii) $T \subseteq A$.

Clearly, every stable model is also a supported model, but the converse may not be the case. We illustrate this on an example.

Example 2.4. Consider a shapes graph (C, T) and a data graph G defined as follows:

$$C = \{\mathsf{Queen} \leftarrow \exists married.\mathsf{King} \vee \exists has.crown,$$
$$\mathsf{King} \leftarrow \exists married.\mathsf{Queen} \vee \exists has.crown\}$$
$$T = \{\mathsf{King}(Franz)\}$$
$$G = \{married(Harry, Meghan), married(Meghan, Harry),$$
$$married(Franz, Sissi), married(Sissi, Franz),$$
$$has(Franz, crown)\}$$

Consider the following shape assignments:

$$A_1 = \{\mathsf{King}(Harry), \mathsf{Queen}(Meghan), \mathsf{King}(Franz), \mathsf{Queen}(Sissi)\}$$
$$A_2 = \{\mathsf{King}(Meghan), \mathsf{Queen}(Harry), \mathsf{King}(Franz), \mathsf{Queen}(Sissi)\}$$
$$A_3 = \{\mathsf{King}(Harry), \mathsf{Queen}(Meghan), \mathsf{King}(Sissi), \mathsf{Queen}(Franz)\}$$
$$A_4 = \{\mathsf{King}(Meghan), \mathsf{Queen}(Harry), \mathsf{King}(Sissi), \mathsf{Queen}(Franz)\}$$
$$A_5 = \{\mathsf{King}(Franz), \mathsf{Queen}(Sissi)\}$$
$$A_6 = \{\mathsf{King}(Sissi), \mathsf{Queen}(Franz)\}$$

The decorated graphs $I_i = G \cup A_i$ with $1 \leq i \leq 6$ all satisfy the condition $[\![\phi_s]\!]^{I_i} = [\![s]\!]^{I_i}$ for each shape name s occurring in C, and hence, each I_i is a supported model of C. The models I_i with $1 \leq i \leq 4$ have no level assignment that fulfills the conditions of Definition 2.3 item (ii), hence they are not stable models. However, I_5 and I_6 are stable models. To show that I_5 is a stable model we give the following level assignment, with $v \in \{Harry, Megan, Franz, Sissi\}$:

$$level(\top, v) = 0 \qquad level(\exists has.crown, Franz) = 0$$
$$level(\text{King}, Franz) = 1 \qquad level(\text{Queen}, Sissi) = 1$$

Note that $level(\phi_s, v) < level(s, v)$ for all $s(v) \in G \cup A_5$ as required. The level assignment for I_6 is similar.

Brave and Cautious validation The above notions of validation only check for the existence of a supported- or stable model, which can be viewed as *brave* validation. Andresel et al. [4] also study *cautious* validation, which aims to validate the targets that are true in every stable model. We recall this notion for both semantics.

Definition 2.5. A data graph G *cautiously validates* a shapes graph (C, T) under the supported (stable) model semantics if $T \subseteq A$ for *every* supported (stable) model $G \cup A$ of C.

We illustrate brave and cautious validation with an example.

Example 2.6. Consider Example 2.4. The target T is contained in the supported models with assignment A_1, A_2 and A_5, but not A_3, A_4 and A_5. Hence, the data graph G bravely, but not cautiously, validates the shapes graph (C, T) under the supported model semantics. For the stable model semantics there exist two models, namely $G \cup A_5$ and $G \cup A_6$. Since $T \subseteq A_5$, but $T \nsubseteq A_6$ the data graph G bravely, but not cautiously validates the shapes graph (C, T) under the stable model semantics.

To ease presentation, we use the following normal form for SHACL constraints.

Definition 2.7 (Normal form for constraints). A constraint is in *normal form* if it has one of the following forms:

$$\text{(NF1) } s \leftarrow \top \qquad \text{(NF2) } s \leftarrow a \qquad \text{(NF3) } s \leftarrow E = E'$$
$$\text{(NF4) } s \leftarrow \neg s' \qquad \text{(NF5) } s \leftarrow s_1 \wedge \cdots \wedge s_n \qquad \text{(NF6) } s \leftarrow \geq_n E.s'$$

It was shown in [4] (Proposition 4.2) that a set of constrains C can be transformed in polynomial time into a normalized set of constraints C' such that for every graph G and target set T, G validates (C, T) iff G validates (C', T) under supported- and stable model semantics. Further, it can be shown that every model I of (C, T) can be transformed into a model I' of (C', T), so that both brave- and cautious validation are preserved.

3 MAGIC SHAPES ALGORITHM FOR SHACL

Now we describe the *Magic Shapes* algorithm for SHACL validation. For simplicity, we first present it for the *positive fragment* of SHACL, that is, constraints in normal form that do not include expressions of the form (NF4). In the next section, we extend it to constraints with unrestricted use of negation. The transformation follows ideas from the *Magic Sets* algorithm for Datalog$^\neg$ in [13].

We first identify the shape names that are *reachable* from a target and the constraints that are relevant for validating it.

Definition 3.1. Given a shapes graph (C, T), where C is positive, we let $reach(C, T)$ be the smallest set of shape names such that:

- if $s(a) \in T$, then $s \in reach(C, T)$, and
- if $s \leftarrow \phi \in C$ and $s \in reach(C, T)$, then every shape name occurring in ϕ is also in $reach(C, T)$.

The set of constraints $C_T = \{s \leftarrow \phi \in C \mid s \in reach(C, T)\}$ is called the *module* of C w.r.t. T.

Example 3.2. Consider a shapes graph (C, T), where C contains:

$$s_1 \leftarrow s_2 \wedge s_3 \qquad s_2 \leftarrow \exists r.s_4 \qquad s_3 \leftarrow s_2$$
$$s_4 \leftarrow \exists r \qquad s_5 \leftarrow s_6 \qquad s_6 \leftarrow \exists r.s_2$$

and $T = \{s_1(a)\}$. Then, $reach(C, T) = \{s_1, s_2, s_3, s_4\}$, resulting in the module $C_T = \{s_1 \leftarrow s_2 \wedge s_3, s_2 \leftarrow \exists r.s_4, s_3 \leftarrow s_2, s_4 \leftarrow \exists r\}$.

The algorithm takes as input a shapes graphs (C, T) and outputs a new optimized (C_{magic}, T) against which data graphs can be validated. The algorithm consists of four main steps, which are summarized in Algorithm 1 and described in more detail below.

Input : SHACL shapes graph (C, T)
Output: Optimized shapes graph (C_{magic}, T).
1 **begin**
2 $C_{\text{generate}} := \{\text{magic\_}s \leftarrow v \mid s(v) \in T\}$;
3 $C_T := \{s \leftarrow \phi \in C \mid s \in reach(C, T)\}$;
4 **foreach** $s \leftarrow \phi \in C_T$ **do**
5 $C_{\text{generated}} := C_{\text{generated}} \cup \{\texttt{Generate}(s \leftarrow \phi)\}$;
6 **end**
7 $C_{\text{modified}} := \{s \leftarrow \text{magic\_}s \wedge \phi \mid s \leftarrow \phi \in C_T\}$;
8 $C_{\text{magic}} := C_{\text{generated}} \cup C_{\text{modified}}$;
9 **return** (C_{magic}, T);
10 **end**

Algorithm 1: Magic Shape Algorithm for positive SHACL

Modularize. The step *modularize* in line 3 extracts the set of constraints relevant for the targets, i.e. the module C_T. Note that this depends on the shape names in T, but not on the targeted nodes.

Create magic seeds. In line 2, constraints of the form magic_$s \leftarrow v$, called *magic seeds*, are created for each shape atom $s(v)$ in T and added to the set C_{generate} of generated constraints.

Example 3.3. Consider again Example 3.2. The algorithm adds the magic seed magic_$s_1 \leftarrow a$ in C_{generate}. Intuitively, this constraint says that the validation of s_1 at a is relevant.

Generate. The module C_T is used to generate *magic constraints* in lines 4-6 through the procedure `Generate`, described in Algorithm 2. In a nutshell, for each $r \in C_T$ of the form $s \leftarrow \phi$ and for each shape name s' in ϕ, it creates a new constraint with `magic_`s in the body and `magic_`s' in the head. If in the constraint two shapes are connected by a path expression E, then E is inverted. Intuitively, these constraints 'mark' the nodes and shapes that are relevant for validating the current target. The constraint `magic_`$s' \leftarrow$ `magic_`s can be read as 'if the s-assignment of a node v is marked as relevant, then the s'-assignment of v is also marked as relevant'.

```
1 function Generate(s ← φ) is
2     if φ = s_1 ∧ ··· ∧ s_n then
3         return
            {magic_s_1 ← magic_s; ...; magic_s_n ← magic_s}
4     if φ =≥_n E.s' then
5         return {magic_s' ←≥_n E⁻.magic_s};
6     else
7         return ∅;
8     end
9 end
```

Algorithm 2: Generation

Example 3.4. In our running example, procedure `Generate` adds four constraints to C_{generate}, namely

$$\text{magic\_}s_2 \leftarrow \text{magic\_}s_1 \qquad \text{magic\_}s_3 \leftarrow \text{magic\_}s_1$$
$$\text{magic\_}s_4 \leftarrow \exists r^-.\text{magic\_}s_2 \qquad \text{magic\_}s_2 \leftarrow \text{magic\_}s_3$$

These constraints together with the magic seeds in Example 3.3 comprise now C_{generate}.

Modification. In line 7, the body of each constraint in the module C_T is enhanced with a magic version of the shape name in its head. Intuitively, this ensures that the resoner does not continue with the validation of shapes assignments at nodes that are not marked as relevant by the magic rules, and restricts models to only the relevant atoms.

Example 3.5. Consider again C_T from Example 3.2. The set of constraints C_{modified} contains:

$$s_1 \leftarrow \text{magic\_}s_1 \wedge s_2 \wedge s_3 \qquad s_2 \leftarrow \text{magic\_}s_2 \wedge \exists r.s_4$$
$$s_3 \leftarrow \text{magic\_}s_3 \wedge s_2 \qquad s_4 \leftarrow \text{magic\_}s_4 \wedge \exists r$$

The output C_{magic} of the magic shapes algorithm consists of the union of the sets $C_{\text{generated}}$ and C_{modified}, see line 8 of Algorithm 1.

Example 3.6. To demonstrate the magic shapes algorithm, consider a data graph $G = \{r(a,a), r(a,c)\}$. When validating G against the shapes graph (C,T) from Example 3.2, there exists a stable model $G \cup A$, with the assignment being

$$A = \{s_1(a), s_2(a), s_3(a), s_4(a), s_5(a), s_6(a),$$
$$s_1(c), s_2(c), s_3(c), s_4(c)\}.$$

The target $s_1(a)$ is in A, and therefore, G validates (C,T). When validating G against the magic shapes graph (C_{magic}, T), then we get a supported model $G \cup M \cup A'$, with

$$M = \{\text{magic\_}s_1(a), \text{magic\_}s_2(a), \text{magic\_}s_3(a), \text{magic\_}s_4(a)\}$$
$$A' = \{s_1(a), s_2(a), s_3(a), s_4(a)\}.$$

Again, it is the case that $s_1(a) \in A'$. Note that the assignment $A' \subseteq A$ contains only 4 shape atoms. Indeed, the module C_T does not contain the constraints regarding s_5 and s_6, which are not reachable from the target, and hence, $s_5(a)$ and $s_6(a)$ are not in A'. Moreover, the set M, obtained from C_{generate} and G, discards the irrelevant magic shape atoms concerning node c, which together with C_{modified}, restrict the assignments of the shape names s_1 to s_4 to the relevant node a only.

4 ALGORITHM TO SUPPORT NEGATION

The presence of arbitrary negation and recursion in SHACL constraints may cause inconsistency with the input data graph, making every target inherently invalid as there cannot exist a stable- or supported model that satisfies the constraints. E.g., for a constraint of the form $s \leftarrow \neg s$, there is no way to assign or not assign a node to the shape name s so that the constraint is satisfied. It is well-known from the database literature [19] that parts of a logic program that may give rise to inconsistency can be caused by this kind of *cyclic dependencies* with an odd number of negated predicates. An analogous situation arises in SHACL, where *odd cycles* may make some assignments invalid. We need to identify such so-called *dangerous* constraints that appear under the scope of negation, and if they are relevant for validating the target, they should participate in the magic algorithm. In this way, the magic shapes graph preserves all valid shape assignments, and only allows validation of inconsistent inputs if the contradictions don't interact with validating the target.

Definition 4.1. [Dangerous constraints] Assume a set of constraints C. The *shape dependency graph* of C is the *marked* directed graph DG_C with an edge from s_1 to s_2 if there is a constraint $s_2 \leftarrow \phi$ in C such that s_1 occurs in ϕ, and the edge (s_1, s_2) is *marked* if $\phi = \neg s_2$. A *cycle* of DG_C is a sequence of nodes $S = n_1, \ldots, n_k$, such that $n_1 = n_k$, each n_i for $1 < i < k$ occurs exactly once in S, and (n_i, n_{i+1}) for $(1 \le i < k)$ is an edge in DG_C. An *odd cycle* in DG_C is a cycle, where an odd number of edges are marked.

A shape name s is *dangerous* in C if

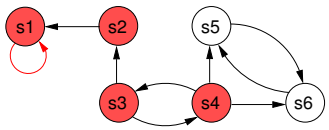
(i) s occurs in an odd cycle of DG_C, or
(ii) s occurs in ϕ for some constraint $s' \leftarrow \phi \in C$ such that s' is dangerous.

We say a constraint $s' \leftarrow \phi$ is dangerous of s' is dangerous.

Example 4.2. Consider the set of shape constraints C:

$$s_1 \leftarrow \neg s_1 \wedge s_2 \qquad s_2 \leftarrow s_3 \qquad s_3 \leftarrow \exists r.\neg s_4$$
$$s_4 \leftarrow \exists r.\neg s_3 \qquad s_5 \leftarrow s_4 \wedge \neg s_6 \qquad s_6 \leftarrow s_4 \wedge \neg s_5$$

This set has one odd cycle involving the shape name s_1. Since s_1 is dangerous and affects s_2, s_2 affects s_3, and s_3 affects s_4, the shape names s_1 to s_4 are dangerous. The figure below shows the shape dependency graph with its dangerous shapes and odd cycle in red.



Extended modules For consistent sets of constraints and data graphs, we want to ensure cautious and brave soundness and completeness, and for inconsistent ones, we want to ensure brave completeness and cautious soundness. For both cautious and brave validation, the completeness property means that if G validates (C,T), then G validates (C_{magic}, T), and soundness means that if G validates (C_{magic}, T), then G validates (C,T). Clearly, inconsistent inputs trivially cautiously validate the shapes graph, and it may be that G bravely validates (C_{magic}, T). Thus brave soundness and cautious completeness cannot be ensured for inconsistent inputs.

In the presence of unrestricted negation in constraints, it may not be sufficient to consider reachability of shape names only from

left to right as for the positive case. This may leave out some relevant dangerous constraint which may be triggered by shape names that are reachable from the target, and which can only be included by propagating from right to left. It is important to detect and appropriately include such constraints in the Magic Shapes algorithm. We illustrate the problem with the following example.

Example 4.3. Consider the data graph $G = \{r(a, a)\}$ and shapes graph (C, T), where $T = \{s_4(a)\}$ and C is the set of constraints from Example 4.2. Note that for simplicity of presentation in this particular example the constraints are not in normal form, but can be easily normalized. There are 2 supported models of C and G, which also coincide with its stable models, namely $I_1 = \{r(a, a), s_4(a), s_5(a)\}$ and $I_2 = \{r(a, a), s_4(a), s_6(a)\}$. Thus G validates (C, T) both bravely and cautiously. Note that $s_3(a)$ does not participate in any model since they are filtered out by the first constraint.

Now let us consider the shapes graph resulting from the Magic Shapes algorithm. The only shape name that is reachable from the target shape name s_4 is s_3, and hence, the module C_T contains only the constraints for these shape names, namely $s_3 \leftarrow \exists r.\neg s_4$ and $s_4 \leftarrow \exists r.\neg s_3$, which will participate in the algorithm. The result is the set C_{magic} consisting of the constraints:

$$\begin{aligned}
\texttt{magic_}s_4 &\leftarrow a,\\
\texttt{magic_}s_3 &\leftarrow \exists r^-.\texttt{magic_}s_4,\\
\texttt{magic_}s_4 &\leftarrow \exists r^-.\texttt{magic_}s_3,\\
s_3 &\leftarrow \texttt{magic_}s_3 \wedge \exists r.\neg s_4,\\
s_4 &\leftarrow \texttt{magic_}s_4 \wedge \exists r.\neg s_3
\end{aligned}$$

The only models of C_{magic} are

$$\begin{aligned}
I_1' &= \{r(a, a), \texttt{magic_}s_3(a), \texttt{magic_}s_4(a), s_3(a)\}, \text{and}\\
I_2' &= \{r(a, a), \texttt{magic_}s_3(a), \texttt{magic_}s_4(a), s_4(a)\}.
\end{aligned}$$

Of these two models, only I_1' contains the target shape atom $s_4(a)$, so G validates (C_{magic}, T) bravely, but not cautiously.

There are also cases where G bravely validates (C_{magic}, T), but not (C, T). For instance, consider again the graph G and the shapes graph (C, T'), where $T' = \{s_3(a)\}$. The module $C_{T'}$ remains the same as C_T, and thus the stable models I_1 and I_2 remain the same. The target T' is neither contained in I_1 nor in I_2, therefore G neither bravely nor cautiously validates the shapes graph (C, T'). When applying the magic shape algorithm on (C, T'), the output C'_{magic} contains the same constraints as above, except for the *magic seed*, which is now $\text{magic\_}s_3 \leftarrow a$. Then the supported and the stable models of C'_{magic} are:

$$\begin{aligned}
I_3' &= \{r(a, a), \text{magic_}s_3(a), \text{magic_}s_4(a), s_4(a)\} \text{ and}\\
I_4' &= \{r(a, a), \text{magic_}s_3(a), \text{magic_}s_4(a), s_3(a)\}
\end{aligned}$$

Now the target T' is contained in I_4', so G bravely validates the shapes graph (C'_{magic}, T') although (C, T') does not.

As illustrated through the above example, on consistent inputs, the current version of the Magic Shapes algorithm for shapes graphs with negation is not cautious-complete and brave-sound. In the above example, the reason for the difference in validation with the original shapes graph is the presence of the second model I_1' which contains s_3. Intuitively, the first constraint of C filters out every model that contain a shape atom over the shape name s_1, and together with the second constraint do not allow also shape atoms over s_3. In fact, the first and second constraint are both *dangerous*.

To address this issue and to ensure both brave and cautious validation (soundness and completeness) for consistent inputs, we extend the notion of reachable shape names and module to take into consideration dangerous constraints. If they are relevant for validating the target, reachability should be propagated also from the body to the head.

Definition 4.4. [Extended reachable set, module] Given a shapes graph (C, T), $exReach(C, T)$ is the set of *extended reachable shape names* in C from shape names in T such that:

(i) if $s(a) \in T$, then $s \in exReach(C, T)$,
(ii) if $s(a) \in T$ and there exists a directed path in DG_C from s to s', then $s' \in exReach(C, T)$, and
(iii) if $s \in exReach(C, T)$, s' is dangerous and there exists a directed path in DG_C from s' to s, then $s' \in exReach(C, T)$.

The set of constraints $C_T = \{s \leftarrow \phi \in C \mid s \in exreach(C, T)\}$ from C is called a *module* of C w.r.t. T.

Complete Algorithm for SHACL with negation We now describe how to adapt the algorithm to take into account the effect of dangerous constraints. The only function that is extended is `Generate`, which will produce more constraints for dangerous constraints. Algorithm 1 will remain the same.

```
1  function Generate(s ← φ) is
2      if φ =≥n E.s' then
3          return magic_s' ←≥n E⁻.magic_s;
4      else
5          foreach shape name s' in φ do
6              return magic_s' ← magic_s;
7          end
8      end
9      if s ← φ is dangerous then
10         if φ =≥n E.s' then
11             return magic_s ←≥n E.magic_s';
12         else
13             foreach shape name s ≠ s_i in φ do
14                 return magic_s ← magic_s_i;
15             end
16         end
17 end
```

Algorithm 3: Generation

Roughly, C_{generate} now contains for each dangerous constraints a set of 'magic' constraints that consider all possible alternations of the shape names in the head and the body of the constraint (see Algorithm 3 lines 9-16). These constraints propagate in assignments the mapping of nodes also from the shape names in the body to the shape name in the head of constraints.

Example 4.5. Consider again Example 4.3. The new C''_{magic} resulting from the complete Magic Shapes algorithm extends the set of constraints C_{magic} from Example 4.3 with the following constraints

generated by the extended procedure `Generate` from Algorithm 3:

$$\begin{aligned}
\texttt{magic_}s_2 &\leftarrow \texttt{magic_}s_1 & \texttt{magic_}s_1 &\leftarrow \texttt{magic_}s_2\\
\texttt{magic_}s_3 &\leftarrow \texttt{magic_}s_2 & \texttt{magic_}s_2 &\leftarrow \texttt{magic_}s_3\\
\texttt{magic_}s_3 &\leftarrow \exists r.\texttt{magic_}s_4 & \texttt{magic_}s_4 &\leftarrow \exists r.\texttt{magic_}s_3\\
s_1 &\leftarrow \texttt{magic_}s_1 \wedge \neg s_1 \wedge s_2 & s_2 &\leftarrow \texttt{magic_}s_2 \wedge s_3
\end{aligned}$$

Note that C''_{magic} does not contain the constraints for s_5 and s_6 since they are not dangerous constraints. Indeed, they are not relevant for validating the target. The unique stable model of C''_{magic} is $I = \{r(a,a), \texttt{magic\_}s_4(a), \texttt{magic\_}s_3(a), \texttt{magic\_}s_2(a), \texttt{magic\_}s_2(a), s_4(a)\}$.

Hence, G both cautiously and bravely validates (C''_{magic}, T) as it does with the input shapes graph (C, T).

5 CORRECTNESS OF THE ALGORITHM

For a given data graph G and a shapes graph (C, T), the Magic Shapes algorithm outputs a new shapes graph (C_{magic}, T), where C_{magic} is composed of C_{generate} and C_{modified}. Recall that the constraints in C_{generate} define the magic version of the adorned shape names and C_{modified} modifies constraints from the input C by adding in the body the magic version of the shape name that is in the head of the constraint. Provided that G is consistent with C, we want to ensure that G validates (C, T) iff G validates (C_{magic}, T). Otherwise, we want to guarantee cautious soundness and brave completeness. To this aim, we start with a simple fact:

Lemma 5.1. *Let G be a data graph and let (C, T) be a shapes graph. Then there exists a unique assignment A such that $G \cup A$ is a stable model of C_{generate}.*

This holds because all the constraints in C_{generate} are positive. Analogously to normal logic programs, the stable model M of C_{generate} is the unique, minimal assignment that is a supported model. Intuitively, M identifies the *relevant universe* of shape atoms for validation, that is, it 'marks' with magic shapes all nodes of the graph that can participate in the validation of the target C_{modify} under both the stable and supported model semantics. We will see that a shapes graph and its magic output coincide on the shape atoms that appear in M.

To show the correctness of the algorithm, we will show that C_{magic} defines a kind of module that is *independent* from the rest of C and G. To this aim we rely on the notion of *ground constraints*, which intuitively instantiate a constraint at a node or a pair of nodes.

Definition 5.2. A *ground constraint* is a tuple (ρ, v), if ρ is a constraint of the form (NF1)-(NF5) and a tuple $(\rho, (v, v'))$, if ρ is a constraint of the form (NF6), where v, v' are nodes from $\mathbf{N}$. We may say (ρ, v) (or $(\rho, (v, v'))$) is the *grounding* of ρ w.r.t. v (or (v, v')).

Next we define the notion of the *grounding* of a set of constraints w.r.t. an input data graph G.

Definition 5.3. Given a data graph G and a set of constraints C, the *grounding* $gr(C, G)$ of C w.r.t. G is a set of ground constraints obtained for each $\rho \in C$, such that

- if ρ is of the form (NF1)-(NF5), then (ρ, v) is in C_{gr} for each $v \in V(G)$,
- if ρ is of the form (NF6), then $(\rho, (v, v'))$ is in C_{gr} for each $v, v' \in V(G)$.

We lift the definition π_s of s in a set of constraints C to ground constraints, and then adapt the definition of supported and stable models to ground constraints as follows.

Definition 5.4. A decorated graph I is a supported model of a set of ground constraints C_{gr} if $[\![\phi_s]\!]^I = [\![s]\!]^I$ for each shape name s occurring in C_{gr} and for each $v \in [\![s]\!]^I$ there exists in C_{gr} a ground constraint (ρ, v) or $(\rho, (v, v'))$ with s in the head of ρ. I is a stable model of C_{gr} if it is a supported model of C_{gr}, and there exists a level assignment such that $level(\phi_s, v) < level(s, v)$ for all $s(v) \in I$.

From the above definitions, the following lemma immediately holds. In particular, this is the case since the grounding $gr(C, G)$ of a set of constraints C w.r.t. a data graph G grounds every constraint in C with every node or every pair of nodes from $V(G)$.

Lemma 5.5. *Let G be a data graph and let C be a set of constraints. Then I is a supported (or stable) model of C iff I is a supported (or stable) model of $gr(C, G)$.*

Independent constraints We now define the notion of *independence* of a set of ground constraints w.r.t. another set of ground constraints, which is similar to the same notion for ground ASP programs defined by Eiter et al. in [11] and to splitting sets defined by Lifschitz et al. in [17].

Definition 5.6. Given a set of ground constraints C_{gr}, the *shape domain* of C_{gr}, written as $\text{sdom}(C_{gr})$, is the set of shape atoms $s(v)$ such that one of the following holds:

- (ρ, v) is in C_{gr} and s occurs in the body or head of ρ, or
- $(\rho, (v, v'))$ is in C_{gr} and s is the head of ρ, or
- $(\rho, (v', v))$ is in C_{gr} and s is in the body of ρ.

Let C_1 and C_2 be sets of ground constraints. We say that C_1 is *independent* of C_2 if it is not the case that there is some shape name s that occurs in the head of some ground constraint (ρ, v) or of the form $(\rho, (v, v'))$ in C_2 such that $s(v)$ occurs in $\text{sdom}(C_1)$.

The above definition roughly states that C_1 is independent of C_2 if each 'ground' shape name that occurs in the head of a ground constraint in C_2 does not occur in any ground constraint (head or body) in C_1, hence the ground constraints in C_1 may have some effect on the ground constraints of C_2, but this is not the case for the other direction.

Theorem 5.7. *Let G be a data graph and let $C_{gr} = gr(C, G)$ be the grounding of C w.r.t. G. Moreover, let C_1 and C_2 be such that $C_{gr} = C_1 \cup C_2$ and C_1 is independent of C_2. Let $M = \text{sdom}(C_1)$. Then, for every supported (or stable) model $I = G \cup A$ of C_{gr} it is the case that $G \cup (A \cap M)$ is a supported (or stable) model of C_1.*

Proof. We show the claim first for the supported model semantics and then for supported models with a level assignment. Let $I = G \cup A$ be a supported model of C_{gr} and let $I_M = G \cup (A \cap M)$. We need to show that I_M is a supported model of C_1, that is: 1) that $[\![\phi_s]\!]^{I_M} = [\![s]\!]^{I_M}$ for each shape name s occurring in C_1, and 2) for each $v \in [\![s]\!]^{I_M}$ there exists in C_1 a ground constraint (ρ, v) or $(\rho, (v, v'))$ with s in the head of ρ. To see the latter, note that $s(v)$

is in M, so there is some constaint with s in the head or the body and v in the corresponding position, as in Definition 5.6. Note that it cannot occur in a body only, since there must be some constraint with s in the head and grounded with v in either of the C_i, but as C_1 is independent from C_2, it cannot be in C_2.

To show 1), we first note that since C_1 is independent from C_2, for each node $v \in [\![s]\!]^{I_M}$ it is the case that all the ground constraints (ρ, v) or $(\rho, (v, v'))$ with s in the head of ρ that appear in C_{gr} will also appear in C_1. Thus there is a unique ϕ_s that is the definiton of s in both C_{gr} and C_1, which is in fact its definiton in C. Now, let s be an arbitrary shape name in C_1 and let v be an arbitrary node in $V(G)$. It is left to show that $v \in [\![\phi_s]\!]^{I_M}$ if and only if $v \in [\![s]\!]^{I_M}$. For the ($\Rightarrow$) direction, $v \in [\![\phi_s]\!]^{I_M}$ implies $v \in [\![\phi_s]\!]^{I}$. Since I is a supported model of C_{gr}, it follows that $v \in [\![s]\!]^{I}$. That is, there is some ground constraint (ρ, v) or $(\rho, (v, v'))$ with s in the head of ρ that appears in C_{gr}, and hence, it also appears in C_1. It follows that $s(v)$ is also in M, and hence also in I_M. For the ($\Leftarrow$) direction, let $s(v) \in I_M$. This implies that $s(v) \in A$ and $s(v) \in M$. The first implies that $v \in [\![\phi_s]\!]^{I}$. Let ϕ be an arbitrary disjunct in ϕ_s such that $v \in [\![\phi]\!]^{I}$. It is left to show that $v \in [\![\phi]\!]^{I_M}$. The claim trivially holds for ϕ as in (NF1), (NF2) or (NF3). If ϕ is of the form $\neg s'$, as in (NF4), then $s'(v)$ must not be in I and hence it is not in A, which implies that $v \in [\![\neg s']\!]^{I_M}$. If ϕ is a conjunction of shape names $s_1, \dots s_n$, as in (NF4), then each $s_i(v)$ is in A. Since $(s \leftarrow s_1 \wedge \cdots \wedge s_n, v)$ is in C_{gr}, then it must also be in C_1, which implies that each $s_i(v)$ is also in M, and hence also in I_M. Finally, for ϕ of the form $\geq_n E.s'$, as in (NF6), reasoning as in the previous case, there must be at least n nodes v_i with an E-path from v such that $s'(v_i)$ in A. Since the grounding is with respect to every node in $V(G)$, for each such v_i there is a ground constraint $(s \leftarrow \geq_n E.s', (v, v_i))$ in C_{gr}, and since C_1 is independent of C_2, each such ground constraint is also in C_1. Hence, by the definition of the shape domain of C_1, $s'(v_i)$ is in M for each such node v_i. From these observations follows that $s'(v_i)$ is in $A \cap M$ for each v_i, which shows that v satisfies $\geq_n E.s'$ in I_M.

For the stable model semantics, it is clear that if I has a level assignment, then $I_M \subseteq I$ also has a level assignment. □

Independence of Magic Constraints We show that to compute the models of a set of constraints C_{magic} resulting from the magic algorithm, it suffices to consider the models of the grounding of the module C_T obtained by grounding every constraint with the nodes of the shapes atoms with 'magic' as prefix from C_{generate}.

Lemma 5.8. *Let G be a data graph, let (C, T) be a SHACL shapes graph and let $G \cup M$ be the stable model of C_{generate}. For each shape name s such that* `magic_s` *appears in M we let $N_s = \{v \mid$* `magic_s`$(v) \in M\}$. *Let $C_{T,gr}$ be the set of ground constraints obtained by grounding every constraint ρ in C_T as follows:*

- *if ρ is of the form (NF1) to (NF5), then $(s \leftarrow \phi, v) \in C_{T,gr}$ for each $v \in N_s$,*
- *if ρ is of the form $s \leftarrow \geq_n E.s'$, that is (NF6), then $(s \leftarrow \phi, (v, v')) \in C_{T,gr}$ for each v, v' with $v \in N_s$ and $v' \in N_{s'}$,*

Then, the following hold under supported and stable model semantics:

(1) if I is a model of C_{magic}, then $I \setminus M$ is a model of $C_{T,gr}$, and
(2) if I is a model of $C_{T,gr}$, then $I \cup M$ is a model of C_{magic}.

Proof. We use the term model instead of supported or stable model. For (1), let I be a model of C_{magic} and let s be an arbitrary shape name occurring in C_{modified}. Then, it is the case that $[\![\phi_s \wedge$ `magic_s`$]\!]^I = [\![s]\!]^I$ and $[\![s]\!]^I \subseteq N_s$. Clearly, $s \leftarrow \phi_s$ is also the definition of s in C_T and by construction, for each $v \in N_s$ and for each constraint $s \leftarrow \phi$ in C_T, there is a ground constraint $(s \leftarrow \phi, v)$ or $(s \leftarrow \phi, (v, v'))$ in $C_{T,gr}$. By the definition of models of ground constraints, it follows that $I \setminus M$ is also a model of $C_{T,gr}$. For (2), let I be a model of $C_{T,gr}$. Then, $[\![\phi_s]\!]^I = [\![s]\!]^I$ and $[\![s]\!]^I \subseteq N_s$. Since N_s contains all the nodes appearing in atoms over `magic_s` in M, follows that $I \cup M$ is a model for C_{generate} and for C_{modified}. For the stable model semantics, the existence of a level assignment for $I \setminus M$ in item (1) is trivial. For (2), it is left to show that if $I = G \cup A$ has a level assignment, then $I \cup M$ has a level assignment. Since $G \cup M$ is the stable model of C_{generate}, there is a level assignment for the atoms with `magic` as prefix such that $level(\phi_{\text{magic}}, v) < level($`magic_s`$, v)$ for every `magic_s`$(v) \in M$, with `magic_s` $\leftarrow \phi_{\text{magic}}$ a constraint in C_{generate}. Let L_G be such an assignment, where the atoms from G are assigned integer 0, and let ℓ be the integer of the highest ranked shape atom in L_G. Furthermore, let L' be a level assignment for I. We define L_M which assigns to each shape atom α in A an integer $i + \ell$, where i is the level assigned to the atom α in L'. Clearly, the level assignment composed of L_G and L_M is as desired. □

The set of ground constraints $C_{T,gr}$ obtained from C_{magic} is independent of the rest of the grounding.

Lemma 5.9. *Let G be a data graph and let $C_{gr} = gr(C, G)$ be the grounding of C w.r.t. G. Then, $C_{T,gr}$ is independent of $C_{gr} \setminus C_{T,gr}$.*

Proof. Towards a contradiction, let s be a shape name that occurs in the head of a constraint ρ with (ρ, v) (or $(\rho, (v, v'))$) in $C_{gr} \setminus C_{T,gr}$ and assume $s(v) \in \text{sdom}(C_{T,gr})$. By Lemma 5.8 follows that `magic_s`(v) is in M, where $G \cup M$ is the stable model of C_{generate}, and there is some constraint with s in the head or body that appears ground in $C_{T,gr}$ with v in the expected position. It follows that the constraint ρ should also appear in C_T since s will also be adorned by the algorithm and by Lemma 5.8 this constraint should also have been grounded in $C_{T,gr}$ with the node v, that is (ρ, v) (or $(\rho, (v, v'))$) should be in $C_{T,gr}$, thus deriving a contradiction to the initial assumption. □

Before we present the main lemma that puts everything together, we need to adapt the notion of dangerous constraints for ground constraints by considering shape atoms rather than shape names.

Definition 5.10. Assume a set of ground constraints C_{gr}. The *ground shape dependency graph* of C_{gr} is a marked directed graph $GDG_{C_{gr}}$, where there is an edge from node $s_1(v)$ to node $s_2(v)$ if there is a ground constraint $(s_2 \leftarrow \phi, v)$ in C such that s_1 occurs in ϕ, an edge from node $s_1(v')$ to node $s_2(v)$ if there is a ground constraint $(s_2 \leftarrow \phi, (v, v'))$ in C_{gr} such that ϕ is of the form $\geq E_n.s_1$, and the edge is marked if $\phi = \neg s_2$. *Cycles* and *odd cycles* of $GDG_{C_{gr}}$ are defined exactly as for DG_C. Let C_{gr} be a set of ground constraints. A shape atom $s(v)$ is *dangerous* if (1) $s(v)$ occurs in an odd cycle of $GDG_{C_{gr}}$, or (2) s occurs in ϕ for some constraint $(s' \leftarrow \phi, v) \in C_{gr}$ or $(s' \leftarrow \phi, (v', v))$ such that s' is dangerous. A ground constraint $(s' \leftarrow \phi, v)$ or $(s' \leftarrow \phi, (v, v'))$ is dangerous if $s'(v)$ is dangerous.

We observe that if a ground constraint (ρ, v) or $(\rho, (v, v'))$ in $gr(C, G)$ is dangerous, then ρ is also dangerous in C.

Lemma 5.11. *Let G be a data graph and let (C, T) a SHACL document. Assume $G \cup M$ is the stable model of C_{generate} and $M' = \{s(v) \mid \texttt{magic\_}s(v) \in M\}$. Then, the following hold under supported- and stable model semantics:*

(1) if $I = G \cup A$ is a model of C, then $I' = G \cup M \cup A'$ is a model of C_{magic}, where $A' = A \cap M'$, and
(2) if C is consistent with G and $I' = G \cup M \cup A'$ is a model of C_{magic}, then there exists an A such that $I = G \cup A$ is a model of C and $A' = A \cap M'$.

Proof. For item (1), we show the claim for supported models and then for models with a level assignment. By Lemma 5.5 it follows that I is a model of C_{gr}. By Lemma 5.9 follows that $C_{T,gr}$ is independent of $C_{gr} \setminus C_{T,gr}$, where C_T is the module of C w.r.t. T and $C_{T,gr}$ is the grounding of C_T w.r.t. the stable model M as defined in Lemma 5.11. The latter and Theorem 5.7, item (1), imply that $G \cup (A \cap M')$ is a model of $C_{T,gr}$, which together with Lemma 5.8, item (2), imply that $I' = G \cup (A \cap M') \cup M$ is a model of C_{magic}. For the stable model semantics, it is left to show that I' has a level assignment if I has a level assignment. This is done analogously to the proof of Lemma 5.8 item (2), arguing that we can combine the level assignments for $G \cup M$ and I into a level assingment for I'.

For item (2), similarly as above, we first show the claim for supported models and argue about the exsitence of a level assignment. Let $I' = G \cup M \cup A'$ be a model of C_{magic}. By Lemma 5.8, item (1), follows that $I' \setminus M$ is a model of $C_{T,gr}$. Let $C_{A'}$ be the set of ground constraints $((s \leftarrow v), v)$ for each v such that $s(v) \in A'$. Since $C_{T,gr}$ is independent from $(C_{gr} \setminus C_{T,gr})$, then we know that we can replace $C_{T,gr}$ by each of its models, and the models of C_{gr} are preserved. That is, if there exists some A such that $I = G \cup A$ is a model of C_{gr} with $A' = A \cap M'$, then I would also a model of $(C_{gr} \setminus C_{T,gr}) \cup C_{A'}$. Clearly, A' would be contained in I. So, it is only left to show that such an A exists, which we argue similarly as in the proof of Theorem 3.4 in [13]: in a nutshell, we show that there is no constraint participating in an odd cycle in $C' = C_{gr} \setminus C_{T,gr}$ that will 'kill' the assignment A' of $C_{T,gr}$.

To this aim, let C_{odd} denote the set of all the ground constraints in C' that are dangerous, that is C_{odd} contains all the ground constraints from C' with a dangerous head that is in an odd cycle, reachable from an odd cycle, or reaches an odd cycle. It follows that the shape domain $\text{sdom}(C_{odd}) \cap \text{sdom}(C' \setminus C_{odd}) = \emptyset$. Moreover, we show that $\text{sdom}(C_{odd}) \cap \text{sdom}(C_{T,gr}) = \emptyset$, which implies that $\text{sdom}(C_{odd}) \cap \text{sdom}(C_{A'}) = \emptyset$. By Definition 5.6, there is no ground constraint (ρ, v) or $(\rho, (v, v'))$ in C_{odd} with s in the head of ρ such that $s(v) \in \text{sdom}(C_{T,gr})$ since $C_{T,gr}$ is independent of C' and C_{odd} is contained in C'. Assume towards a contradiction that there is a ground constraint (ρ, v) or $(\rho, (v', v))$ in C_{odd} with s in the body of ρ such that $s(v) \in \text{sdom}(C_{T,gr})$. First, we observe that since such a ground constraint in C_{gr} is dangerous, then ρ must also also be dangerous in C, and by definition, the shape name s occurring in the body of ρ is also dangerous. Since $s(v) \in \text{sdom}(C_{T,gr})$, then by Definition 5.6, there is a constraint ρ' such that at least one of the following holds:

- (ρ', v) is in $C_{T,gr}$ and s occurs in the body or head of ρ', or
- $(\rho', (v, v'))$ is in $C_{T,gr}$ and s is the head of ρ', or
- $(\rho', (v', v))$ is in $C_{T,gr}$ and s is in the body of ρ'.

Thus ρ' contains a dangerous shape name s in the body or in the head. Hence, by Definition 4.4, ρ should also be present in the module C_T and by Definition 5.2, the ground constraint (ρ, v) or $(\rho, (v', v))$ should also be contained in $C_{T,gr}$ and not in C_{odd}, deriving a contradiction. From the above observations it follows that the shape domain of C_{odd} has no intersection with the shape domain of the rest of the constraints in C_{gr}. Since C_{gr} is consistent, it follows that there exists a model of C_{odd}. Let $G \cup A_{odd}$ be such a model. Note that the rest of the set of ground constraints, that is $(C_{gr} \setminus (C_{T,gr} \cup C_{odd})) \cup C_{A'}$ does not contain any odd cycles. It is known from answer set programming that programs that do not contain any odd cycle are consistent [10], that is they admit a model. This result can be lifted to SHACL by results from Andresel et al. in [4], which showed that a SHACL shapes graph can be translated into a logic program with negation that preserves validation under the stable and supported model semantics. These transformation does not change the parity of negation in cyclic dependencies of shape names. That is, a shape name s appears in a cycle with an odd number of negative edges if and only if the corresponding unary atom over s in targeted program appears in a cycle with an odd number of negative edges. So, as $(C_{gr} \setminus (C_{T,gr} \cup C_{odd})) \cup C_{A'}$ has no odd cycles, we know it has a model. Let $G \cup A' \cup A_{rest}$ be such a model. It then follows that $G \cup A' \cup A_{rest} \cup A_{odd}$ is a model of C_{gr}. By Lemma 5.5 and since $T \subseteq I'$, it follows that the desired model I of C is $I = G \cup A$ where $A = A' \cup A_{rest} \cup A_{odd}$. Either A_{odd} nor A_{res} have any atoms from M', so $A' = A \cap M'$. This finishes the proof for the supported model semantics.

For the stable model semantics, it remains to argue that there is a level assignment for I, which is not hard to do, since we know there are level assingment exists for each of $G \cup A_{rest}$, $G \cup A_{odd}$ and $G \cup A$. As A_{rest} and A_{odd} are disjoint, and they are both disjoint from A', we can combine them into a level assignment for I. □

Intuitively, if we ignore the 'magic' prefix and the shape atoms with the 'magic' prefix, the above lemma states that, for consistent inputs, the set of models of G and C restricted to the shape atoms resulting from C_{generate} is equivalent to the set of the models of G and C_{magic}. For inconsistent inputs, there may exist a model of G and C_{magic} even if there is no assignment that satisfies all the constraints in C. This gives us the desired correctness result:

Theorem 5.12. *Let G be a data graph and let (C, T) be a shapes graph. The following hold under supported and stable model semantics:*

(1) If G bravely *validates (C, T), then G* bravely *validates (C_{magic}, T).*
(2) If G cautiously *validates (C_{magic}, T), then G* cautiously *validates (C, T).*
(3) If C is consistent with G, then G validates (C_{magic}, T) iff G validates (C, T), under both brave *and* cautious *validation.*

Proof. We show the proof for stable models; it is identical for supported models. For (1), suppose $I = G \cup A$ is a stable model of C such that $T \subseteq A$. By Lemma 5.11, item 1, it follows that there is an I' of the form $G \cup M \cup A'$ that is a model of C_{magic}, where $G \cup M$ is the stable model of C_{generate} and $A' = A \cap M'$. It is left to show that $T \subseteq A'$. This follows by construction of C_{generate} which contains a

magic seed for each shape atom in the target, and thus, for every $s(v) \in T$, $\texttt{magic\_}s(v)$ is in M, and hence, $s(v) \in M' \cap A$.

For (2), the claim trivially holds for the case where C is not consistent with G. Otherwise, assume C is consistent with G and suppose G cautiously validates C_{magic}. Then, either (a) C_{magic} is not consistent with G, which by Lemma 5.11 item (1), it follows that C is not consistent with G, and thus the claim follows, or (b) C_{magic} is consistent with G and $T \subseteq I'$ for each stable model I' of C_{magic}. For (b), let $I' = G \cup M \cup A'$ be an arbitrary stable model of C_{magic} such that $T \subseteq A'$. By Lemma 5.11 item (2), there is a stable model $I = G \cup A$ of C that contains A', which implies $T \subseteq A$.

For (3), that C is consistent with G implies C_{magic} is consistent with G. It is left to show (i) if G *bravely* validates (C_{magic}, T), then G bravely validates (C, T), and (ii) if G *cautiously* validates (C, T), then G cautiously validates (C_{magic}, T). Since there is a one to one correspondence between stable models for C and for C_{magic}, arguing as above using Lemma 5.11, item 1 and 2, the claim easily follows for both brave and cautious validation. □

6 INCONSISTENCY-TOLERANT SEMANTICS

As has been observed in previous sections, the Magic Shapes algorithm may output a shapes graph (C_{magic}, T) which is consistent with the data graph although the input shapes graph (C, T) is not. This suggests that the constraints and the part of the data graph causing the inconsistency are not relevant for validating the target. Such situations in the context of SHACL may arise very often due to the large RDF triplestores, which may inevitably, have faulty facts. The 2-valued semantics that we have considered until now, which requires the existence of a global assignment, may be too restrictive in this setting. Corman et al. [8] address this drawback of the 2-valued supported model semantics by proposing the notion of *faithful assignments*, which may leave some shape atoms as *undefined*. We propose a similar *inconsistency-tolerant* semantics for the stable- and supported model semantics, based on the models of our magic transformation. To this aim, we first define a new notion of assignments which may contain negated shape atoms.

Definition 6.1. A *literal assignment* A for a graph G is a set of shape atoms of the form $s(v)$ or $\neg s(v)$ where $s \in \mathbf{S}$ and $v \in V(G)$.

If for a shape name s appearing in C and a node $v \in V(G)$ we have $\{s(v), \neg s(v)\} \cap A = \emptyset$, then we say that $s(v)$ is undefined in A.

Definition 6.2. Assume a set of shape constraints C and a literal assignment A for G. Then, $I = G \cup A$ is called a *3-valued supported model* of C if for each $v \in V(G)$ and for each constraint $s \leftarrow \phi$ in C:

- if $s(v) \in I$, then $v \in [\![\phi]\!]^I$, and
- if $\neg s(v) \in I$, then $v \notin [\![\phi]\!]^I$.

If there is a level assignment for I, then I is *3-valued stable model* of C. A graph G *validates* a shapes graph (C, T) under *the 3-valued supported model semantics* (or 3-valued stable model semantics) if there exists a 3-valued supported model (or 3-valued stable model) I of C such that $T \subseteq A$.

Note that each supported (or stable) model $G \cup A$ of a set of constraints C can be seen as a 3-valued supported (or stable) model $G \cup A'$ where no shape atom remains undefined: simply let A' extend A with the negated version of each shape atom $s(v) \notin A$ with s a shape name appearing in C and v a node in $V(G)$.

While 3-valued validation is highly desirable, to our knowledge no algorithm that allows to decide the existence of a 3-valued stable model for a given (C, T) that may contain unrestricted recursion and negation had been proposed until now, neither under the supported nor under the stable semantics. Our magic transformation, however, is an effective way to do 3-valued validation. Indeed, we can see the models of the magic as 3-valued models where only the *relevant atoms* identified by C_{generate} are defined to be true or false, and where all the remaining atoms, which play no role in the validation of the target, are left undefined.

Lemma 6.3. *Let G be a data graph, let C be a shapes graph, and let $I' = G \cup M \cup A'$ be a supported (stable) model of C_{magic}, where $G \cup M$ is the stable model of C_{generate}. Let $M' = \{s(v) \mid \texttt{magic\_}s(v) \in M\}$. Then, $I = G \cup A$ is a 3-valued supported (stable) model of C, where*

$$A = A' \cup \{\neg s(v) \mid s(v) \in M' \setminus A'\}$$

Proof. In order to show that $I = G \cup A$ is a 3-valued supported model we need to show that (i) if $s(v) \in I$, then $v \in [\![\phi]\!]^I$, and (ii) if $\neg s(v) \in I$, then $v \notin [\![\phi]\!]^I$. Let $I' = G \cup M \cup A'$ be a supported model of C_{magic}, it follows that $[\![s]\!]^{I'} = [\![\texttt{magic\_}s]\!]^{I'} \cap [\![\phi]\!]^{I'}$ for each constraint $s \leftarrow \texttt{magic\_}s \wedge \phi \in C_{\text{magic}}$. Note that the evaluation of shape expressions is not affected by additional negated shape atoms (see Table 1), hence the equality $[\![\phi]\!]^I = [\![\phi]\!]^{I'}$ holds. For positive shape atoms $s(v) \in I$, we know that $s(v) \in A'$, which implies that $v \in [\![\phi]\!]^I$. For the negated shape atoms of the form $\neg s(v) \in A$, we know that $s(v)$ is in M' but not in A', which implies that $v \notin [\![\phi]\!]^I$.

To prove it for the 3-valued stable model semantics it is left to show that I has a level assignment such that for all $s(v) \in I$ it holds that $level(\phi_s, a) < level(s, a)$. Let I' be a stable model of C_{magic}, then it has a level assignment where $level(\texttt{magic\_}s \wedge \phi_s) = max(\{level(\texttt{magic\_}s), level(\phi_s)\}) < level(s, v)$ for all $s(v) \in I'$. From this observation follows that $level(\phi_s) < level(s, v)$ for all $s(v) \in I'$. By construction of I every $s(v)$ in I is also in I', therefore for all $s(v) \in I$ it holds that $level(\phi_s) < level(s, v)$. □

Theorem 6.4. *Consider a data graph G and a shapes graph (C, T). If G validates (C_{magic}, T) under the supported (stable) model semantics, then G validates (C, T) under the 3-valued supported (stable) model semantics.*

Proof. Let $I = G \cup M \cup A'$ be a stable model of C_{magic} and $T \subseteq A$. Then G is valid against a shapes graph (C, T) under the 3-valued supported model semantics (or stable model semantics) if (i) I is a 3-valued supported- or stable model and (ii) $T \subseteq A$. By Theorem 6.3 (i) holds, since $G \cup A$ is a 3-valued supported model (or stable model) of C, where $A = A' \cup \{\neg s(v) \mid s(v) \in M' \setminus A'\}$ and $M' = \{s(v) \mid \texttt{magic\_}s(v) \in M\}$, and (ii) holds by assumption. □

The converse does not hold, as we show in the following example. Our magic technique provides a stronger inconsistency-tolerant validation than arbitrary 3-valued models.

Example 6.5. Consider (C, T) and G, where $C = \{s \leftarrow s' \wedge \neg s, s' \leftarrow \exists r\}$, $T = \{s'(a)\}$ and $G = \{r(a, a)\}$. The set $G \cup \{s'(a)\}$ is a 3-valued stable and supported model that validates the target, but it is not a 2-valued stable or supported model of C_{magic}. Since s occurs in an odd cycle, the constraint $\rho = s \leftarrow s' \wedge \neg s$ is dangerous, and since s' occurs in its body, by Definition 4.4, ρ also occurs in

Table 2: Constraints used in our experiments. We use $=_n r$ to abbreviate $\geq_n r \wedge \leq_n r$, and $\exists r$ to abbreviate $\exists r.\top$.

$C_1 = \{$ Actor $\leftarrow$ Person $\wedge$ ($\exists starring^-$.Movie $\vee$ $\exists occupation.actor$) Famous $\leftarrow$ $\exists knownFor$

Director $\leftarrow$ Person $\wedge$ $\exists director^-$.Movie AwardWinning $\leftarrow$ $\exists award$

Location $\leftarrow$ $\exists country$ Anarchy $\leftarrow$ Location $\wedge$ $\neg\exists leaderTitle$

Movie $\leftarrow=_1$ *imdbId* $\wedge$ $\exists starring$.Actor LivingLanguage $\leftarrow$ $\exists spokenIn$.Location

TranslatedMovie $\leftarrow$ Movie$\wedge \geq_2$ *language*.LivingLanguage Musician $\leftarrow$ Person $\wedge$ $\exists instrument$

Person $\leftarrow=_1$ *birthPlace*.Location Employee $\leftarrow$ $\exists employer$

WorkingPerson $\leftarrow$ Person $\wedge$ Employee Parent $\leftarrow$ Person $\wedge$ $\exists child$

WorkingClass $\leftarrow$ Person $\wedge$ $\exists child^-$.WorkingParent WorkingParent $\leftarrow$ Parent $\wedge$ WorkingPerson $\}$

$C_2 = (C_1 \setminus \{\text{Employee} \leftarrow \exists employer\}) \cup \{\text{Employee} \leftarrow \exists employer.\neg\text{Employee}\}$

Table 3: Results of experiments with the SHACL-ASP validator for C_1 and C_2, and targets T over single nodes or classes with a number of nodes specified in the second column. For C_1, the column 'valid$_{2,3}$' indicates the number of valid nodes under both 2- and 3-valued semantics (the same before and after magic). For C_2, the 2-valued semantics always fails, and 'valid$_3$' shows the number of targeted nodes that are validated under 3-valued semantics using magic. Evaluation times before magic t and after magic t_m are measured in seconds. For each $i \in \{1, 2\}$, $|C_{im}|$ is the number of constraints after magic and N_{im} is the number of graph nodes appearing in the magic shape names in the model of $C_{i\text{generate}}$.

Targets		C_1					C_2				
T	nodes	valid$_{2,3}$	t	t_m	$\lvert C_{1m}\rvert$	N_{1m}	valid$_3$	t	t_m	$\lvert C_{2m}\rvert$	N_{2m}
Musician(*mozart*)	1	0	783	243	6	1	0	860	231	6	1
Musician(*Actor*)	4.872	0	958	253	6	6908	0	957	240	6	6908
Musician(*Person*)	2267445	5248	894	410	6	2396852	5248	936	407	6	2396852
Actor(*cameron*)$^{(1)}$	1	1	768	384	8	3	1	894	404	8	3
Actor(*Actor*)	4872	614	905	450	8	253606	614	929	441	8	253606
Actor(*Person*)	2267445	16645	975	620	8	2567841	16645	933	642	8	2567841
Movie(*Film*)	143121	2449	882	474	8	302030	2449	999	448	8	302030
TranslatedMovie(*Film*)	143121	5	849	491	12	303021	5	905	508	12	303021
Employee(*bill*)	1	1	868	42	2	1	nm	936	45	4	3
Employee(*mark*)	1	0	785	39	2	1	0	831	43	4	1
Employee(*Actor*)	4872	6	825	42	2	4872	6	977	44	4	4879
Employee(*Person*)	2267445	9290	868	86	2	2267445	nm	880	145	4	2273847
WorkingPerson(*bill*)	1	1	846	241	8	2	nm	981	259	10	2
WorkingPerson(*mark*)	1	0	795	234	8	2	0	879	249	10	2
WorkingPerson(*Actor*)	4872	2	864	250	8	6908	2	997	294	10	6915
WorkingPerson(*Person*)	2267445	2959	906	410	8	2396852	nm	970	488	10	2403194
WorkingClass(*bill*)	1	0	734	258	14	2	0	890	283	16	2
WorkingClass(*mark*)$^{(2)}$	1	1	822	291	14	7	1	942	285	16	7
WorkingClass(*Actor*)	4872	0	906	278s	14	7055	0	979	313	16	7056
WorkingClass(*Person*)	2267445	40	887	445	14	2396883	40	985	588	16	2397188

C_T, and hence, the constraint $s \leftarrow \texttt{magic\_}s \wedge s' \wedge \neg s$ will be in the output of the magic algorithm. Since there is no consistent way to assign or not assign node a to shape name s, there is no 2-valued stable or supported model of C_{magic}.

7 IMPLEMENTATION AND EXPERIMENTS

We implemented a prototype of the Magic Shapes algorithm and analysed its effect on the performance of SHACL-ASP, the only existing validator for full SHACL. The source code of the prototype and all files used for the experiments are available online.[2]

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 SHACL-ASP validator with an Apache Jena TDB[3] as an RDF triple store to access the data graph.

Data Graph. The data for the experiments was obtained from DBPedia (latest-core, version 2021-12)[4]. More precisely, we used the datasets 'PersonData', 'Instance Types', 'Labels', 'Mappingbased Literals' and 'Mappingbased Objects'. The datasets are in Turtle syntax and contain about 119 million triples (15.4 GB). The SHACL-ASP validator extracts from this graph all triples that mention the data predicates in the shapes graphs, and transforms them into a set G_{asp} of ASP facts. For our test cases, G_{asp} has almost 7 million facts and 4 million constants (graph nodes). To assess our inconsistency-tolerant validation, we added a few facts to G_{asp}. Specifically, we added *employer*(*bill*, *bob*), *employer*(*bob*, *jim*), and

[2] https://github.com/biziloehnert/magicSHACL.

[3] https://jena.apache.org/documentation/tdb/

[4] https://www.dbpedia.org/resources/latest-core/

employer(*jim*, *bill*) to create an *employer* cycle over three individuals, we also added a *birthPlace* for each of them.

Shapes Graph. We created two sets of shape constraints describing persons and their professions, in the spirit of the test cases in [7]. We show them—in abstract syntax—in Table 2. Note that C_2 can result in inconsistency if the data graph has cycles over the relation *employer*. C_1 and C_2 were combined with different targets to obtain 40 different shapes graphs. We are using *class targets*: the target $s(C)$ is a shortcut for the set of targets containing $s(v)$ for each node v that has type C in G. The number of nodes in the graph that are targeted in our shapes graphs ranges from one to over two million, see the second column of Table 3.

Results. Table 3 summarizes the results of our experiments. The average time (in seconds) of running the validator five times on the original shapes graph and five times on its magic variant for each test case is shown in the t and t_{m} columns. The Magic Shapes algorithm improves the performance of the DLV validator significantly, by at least 30% in our test cases.

The constraints in C_1 are recursive: there is a cycle between shape names Actor and Movie. As C_1 is always consistent–the mild form of negation in C_1 does not break consistency–the number of targeted nodes that are validated under the 2- and 3-valued semantics coincides, independently of the magic. Their total can be seen in the column '$\mathsf{valid}_{2,3}$'.

In contrast, the additional constraint with negation in C_2 is dangerous, as it creates a marked self-loop to the dangerous shape name Employee in the dependency graph DG_{C_2} of C_2. In fact, C_2 is inconsistent with G_{asp} independently of any targets T: due to the mentioned *employer* cycle, there is no way to neither assign nor omit Employee for each of *bill*, *bob* and *jim*, and there is no stable- or supported model. As a consequence, none of the targeted nodes can be validated with the 2-valued semantics. However, using magic, we can validate under the 3-valued semantics all expected nodes for the 16 targets that are not related to these individuals being or not employees, see column 'valid_3'. The magic versions of C_2 over the shape names Musician, Actor, Movie, or TranslatedMovie coincide with the corresponding versions of C_1, and they are positive. This is reflected in the results in Table 3, which are identical, and only t and t_m may be slightly higher for C_2 as it has an additional constraint. For the rest of the targets, the dangerous constraint participates in $C_{2\mathsf{m}}$ (hence there are two more constraints than in $C_{1\mathsf{m}}$, one for $C_{2\mathsf{generate}}$ and one for $C_{2\mathsf{modified}}$. We remark that the constraint for Employee will not participate in the magic version in $C_{1\mathsf{generate}}$, but the new version will be in $C_{2\mathsf{generate}}$. This explains why N_{2m} may be slightly larger than N_{1m}. However, the dangerous constraint will only cause inconsistency if *bill*, *bob*, or *jim* is adorned with magic_Employee by $C_{2\mathsf{generate}}$. In our tests, this is the case for Employee(*bill*), Employee(*Person*), WorkingPerson(*bill*), and WorkingPerson(*Person*). For the rest of the targets in Table 3, the dangerous constraint will behave as the positive constraint for Employee in C_1, validating the same number of nodes.

Finally, we observe that in some cases recursion may eliminated by the magic algorithm (e.g., for C_1 with a Musician target, since the shape names responsible for recursion are not reachable), or negation (e.g., for C_2 with Musician, Actor, Movie, and TranslatedMovie targets). In principle, this could enable the use other validators for SHACL fragments, like SHACL2SPARQL and TRAV-SHACL.[5]

Table 4: Experiments on reduced data graphs. The times in the first colum are for validating the original C_i over G_{asp}, and in the second column for the magic version $C_{i_{\mathsf{m}}}$ over $G_{i\mathsf{m}}$.

	C, G_{asp}	$C_{\mathsf{m}}, G_{i\mathsf{m}}$	$N_{i\mathsf{m}}$	$\lvert G_{i\mathsf{m}}\rvert$
C_1,(1)	768s	0.13s	3	461
C_1,(2)	822s	1.87s	7	26069
C_2,(2)	942s	2.31s	7	26069

Unfortunately, the current SHACL-ASP does not easily allow us to fully leverage the fact that the subgraph identified as relevant may be several order of magnitudes smaller, and it loads the full G_{asp} with its millions of nodes even when only a handful of them are relevant in the validation. To explore the potential for further reducing the data graph, we hand-picked three shape graphs with single nodes as targets, and dropped from G_{asp} all facts not mentioning nodes in N_{im}. Even though the resulting $G_{i\mathsf{m}}$ may still contain thousands of facts (since for example, some locations may occur in thousands of birth place facts), the execution times dropped by a couple of orders of magnitude. This observation suggests that it would be promising to optimize SHACL-ASP to work with a better approximation of $G_{i\mathsf{m}}$.

8 CONCLUSIONS AND OUTLOOK

Adapting the Magic Set technique, we have proposed a *magic* transformation for unrestricted SHACL validation. It takes as input a shapes graph (C, T) and produces a new (C_{magic}, T). The new C_{magic} not only discards shape constraints that are irrelevant for validating the target, but also identifies the relevant parts of the data graph, thus enabling a target-oriented validation. The new shapes graph can be equivalently used for validation on every graph that is consistent with C. In case of inconsistency, C_{magic} may still admit partial, target-oriented validation. To our knowledge, no practical approaches for such inconsistency tolerant validation were available to date. We implemented our magic transformation and ran experiments with SHACL-ASP, the only SHACL validator that can support unrestricted recursion and negation, and showed that the simpler shapes graphs produced by the Magic Shapes algorithm are evaluated significantly more efficiently than the original ones.

We are currently exploring possible ways to exploit the Magic Shapes technique in a more efficient and robust SHACL validator. Specifically, we want to develop an improved SHACL validator which does not require to load huge graphs as sets of facts, but instead uses a SPARQL end-point (in the style of SHACL2SPARQL and TRAV-SHACL) to access only the potentially much smaller data graph that the magic algorithm identifies as relevant.

ACKNOWLEDGMENTS

This work was funded in whole or in part by the Faculty of Informatics of TU Wien, the Vienna Business Agency, the Austrian Science Fund (FWF) projects P30360 and P30873, and the FWF and netidee SCIENCE project T 1349-N.

[5]We could not run the output of magic on these prototypes since their parsers appear to be incomplete and do not support constraints like $s \leftarrow s'$.

REFERENCES

[1] Weronika T. Adrian, Mario Alviano, Francesco Calimeri, Bernardo Cuteri, Carmine Dodaro, Wolfgang Faber, Davide Fuscà, Nicola Leone, Marco Manna, Simona Perri, Francesco Ricca, Pierfrancesco Veltri, and Jessica Zangari. 2018. The ASP System DLV: Advancements and Applications. *Künstliche Intell.* 32, 2-3 (2018), 177–179. https://doi.org/10.1007/s13218-018-0533-0

[2] Mario Alviano and Wolfgang Faber. 2011. Dynamic magic sets and super-coherent answer set programs. *AI Communications* 24, 2 (2011), 125–145.

[3] Mario Alviano, Wolfgang Faber, Gianluigi Greco, and Nicola Leone. 2012. Magic sets for disjunctive datalog programs. *Artificial Intelligence* 187 (2012), 156–192.

[4] Medina Andresel, Julien Corman, Magdalena Ortiz, Juan L. Reutter, Ognjen Savkovic, and Mantas Šimkus. 2020. Stable Model Semantics for Recursive SHACL. In *Proc. of The Web Conference 2020*. ACM, 1570–1580. https://doi.org/10.1145/3366423.3380229

[5] Franz Baader, Ian Horrocks, Carsten Lutz, and Ulrike Sattler. 2017. *An Introduction to Description Logic*. Cambridge University Press. http://www.cambridge.org/de/academic/subjects/computer-science/knowledge-management-databases-and-data-mining/introduction-description-logic?format=PB#17zVGeWD2TZUeu6s.97

[6] F. Bancilhon, D. Maier, Y. Sagiv, and J. Ullman. 1985. Magic sets and other strange ways to implement logic programs (extended abstract). In *PODS '86*.

[7] Julien Corman, Fernando Florenzano, Juan L. Reutter, and Ognjen Savkovic. 2019. Validating Shacl Constraints over a Sparql Endpoint. In *ISWC*. Springer. https://doi.org/10.1007/978-3-030-30793-6_9

[8] Julien Corman, Juan L. Reutter, and Ognjen Savkovic. 2018. Semantics and Validation of Recursive SHACL. In *Proc. of ISWC'18*. Springer. https://doi.org/10.1007/978-3-030-00671-6_19

[9] Chiara Cumbo, Wolfgang Faber, Gianluigi Greco, and Nicola Leone. 2004. Enhancing the Magic-Set Method for Disjunctive Datalog Programs. In *Logic Programming, 20th International Conference, ICLP (Lecture Notes in Computer Science)*, Bart Demoen and Vladimir Lifschitz (Eds.). Springer. https://doi.org/10.1007/978-3-540-27775-0_26

[10] Phan Minh Dung. 1992. On the Relations between Stable and Well-Founded Semantics of Logic Programs. *Theor. Comput. Sci.* 105, 1 (1992), 7–25. https://doi.org/10.1016/0304-3975(92)90285-N

[11] Thomas Eiter, Georg Gottlob, and Heikki Mannila. 1997. Disjunctive Datalog. 22, 3 (1997), 364–418. https://doi.org/10.1145/261124.261126

[12] Thomas Eiter, Giovambattista Ianni, and Thomas Krennwallner. 2009. Answer Set Programming: A Primer. In *Reasoning Web. Semantic Technologies for Information Systems, 5th International Summer School 2009, Brixen-Bressanone, Italy, August 30 - September 4, 2009, Tutorial Lectures (Lecture Notes in Computer Science)*, Vol. 5689. Springer, 40–110. https://doi.org/10.1007/978-3-642-03754-2_2

[13] Wolfgang Faber, Gianluigi Greco, and Nicola Leone. 2007. Magic Sets and their application to data integration. *J. Comput. Syst. Sci.* (2007), 584–609. https://doi.org/10.1016/j.jcss.2006.10.012

[14] Mónica Figuera, Philipp D. Rohde, and Maria-Esther Vidal. 2021. Trav-SHACL: Efficiently Validating Networks of SHACL Constraints. In *WWW*. ACM / IW3C2, 3337–3348. https://doi.org/10.1145/3442381.3449877

[15] José Emilio Labra Gayo, Eric Prud'hommeaux, Iovka Boneva, and Dimitris Kontokostas. 2017. *Validating RDF Data*. Morgan & Claypool Publishers. https://doi.org/10.2200/S00786ED1V01Y201707WBE016

[16] Martin Leinberger, Philipp Seifer, Tjitze Rienstra, Ralf Lämmel, and Steffen Staab. 2020. Deciding SHACL Shape Containment Through Description Logics Reasoning. In *Proc. of ISWC 2020 (Lecture Notes in Computer Science)*, Vol. 12506. Springer, 366–383. https://doi.org/10.1007/978-3-030-62419-4_21

[17] Vladimir Lifschitz and Hudson Turner. 1999. Splitting a Logic Program. (01 1999).

[18] Inderpal Singh Mumick, Sheldon J. Finkelstein, Hamid Pirahesh, and Raghu Ramakrishnan. 1990. Magic is Relevant. In *Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data*, Hector Garcia-Molina and H. V. Jagadish (Eds.). ACM Press, 247–258. https://doi.org/10.1145/93597.98734

[19] Christos H. Papadimitriou and Mihalis Yannakakis. 1997. Tie-Breaking Semantics and Structural Totality. *J. Comput. Syst. Sci.* 54 (1997), 48–60.

[20] Paolo Pareti, George Konstantinidis, Fabio Mogavero, and Timothy J. Norman. 2020. SHACL Satisfiability and Containment. In *Proc. of ISWC 2020*. Springer. https://doi.org/10.1007/978-3-030-62419-4_27

[21] Kenneth A. Ross. 1994. Modular Stratification and Magic Sets for Datalog Programs with Negation. *J. ACM* 41, 6 (1994). https://doi.org/10.1145/195613.195646