Well-founded Semantics for Recursive SHACL

Adrian Chmurovič[1], Mantas Šimkus[1,2]

[1]Faculty of Informatics, TU Wien, Austria
[2]Department of Computing Science, Umeå University, Sweden

Abstract

W3C has recently introduced SHACL as a new standard for integrity constraints on RDF graphs. Unfortunately, the standard defines the semantics of *non-recursive* constraints only, which has spurred recent research efforts into finding a suitable, mathematically crisp semantics for constraints with cyclic dependencies. To this end, Corman et al. [1] introduced a semantics related to *supported models* known in logic programming, while Andreşel et al. [2] presented a semantics based on *stable models* known in *Answer Set Programming (ASP)*. In this paper, we argue that recursive SHACL can be naturally equipped with a semantics inspired in the *well-founded semantics* for (recursive) DATALOG queries with default negation. This semantics is not only intuitive, but it is also computationally tractable, unlike the previous proposals. To draw a connection with the classic definition of well-founded semantics in logic programming, we provide a simple translation of recursive SHACL under the well-founded semantics into propositional logic programs under the well-founded semantics. This translation is not only of theoretical interest: It can be efficiently implemented using the functionality of standard RDF triple stores (specifically, by issuing a small number of SPARQL queries), followed by an efficient evaluation using an existing engine for computing the well-founded model of a propositional logic program.

Keywords

graph-structured data, integrity constraints, SHACL, RDF, well-founded semantics, answer set programming

1. Introduction

Graph-structured data is becoming increasingly prominent and popular, mainly because it does not require the development of rigid database schemas. This is important in emerging scenarios where data has complex structure, where it may be highly incomplete, or where its structure evolves over time, which makes the development of a fixed database schema challenging. This has spurred research into topics like *Knowledge Graphs*, *Property Graphs*, *Graph Databases*, *RDF graphs*, etc. The flexible nature of graph-structured data does not mean we are uninterested in the quality and the structural integrity of the stored data. In traditional relational databases, the so-called *integrity constraints* provide means to enforce a strict structure and maintain quality of stored information, and we would like to have similar tools for graph-structure data. However, this is not easy: we need to strike a good balance between the power to assert some control over the structure of data, and being relaxed enough to not negate the benefits of the graph-based data model.

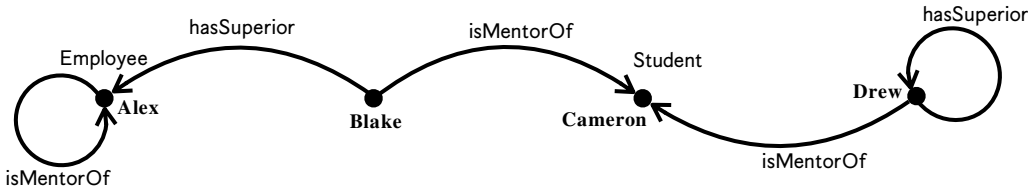
Datalog 2.0 2022: 4th International Workshop on the Resurgence of Datalog in Academia and Industry, September 05, 2022, Genova - Nervi, Italy
simkus@cs.umu.se (M. Šimkus)

To address the above challenge, W3C has recently introduced SHACL[1] as a new standard for expressing constraints on RDF graphs (we refer the reader to [3] for an excellent tutorial on SHACL). Constraints in SHACL are specified using validation rules, which have features reminiscent of logic programming and concept expressions expressed in *Description Logics*. Unfortunately, the SHACL standard only defines the semantics of *non-recursive* SHACL constraints, while the semantics of SHACL with cyclic dependencies, or *recursion*, was intentionally left unspecified: it is currently up to the developers of SHACL validators to come up with ad-hoc solutions to handle recursion. This has spurred recent research efforts into finding a suitable, mathematically crisp semantics for recursive SHACL constraints. Corman et al. have introduced a semantics that is related to the notion of *supported models* known in logic programs, while Andreşel et al. have explored a semantics based on *stable models* known in Answer Set Programming (ASP) [1, 2]. The supported model semantics of Corman et al. has two weaknesses: (a) the problem of deciding validity is intractable, (b) it allows for unfounded (self-supported) justifications of inferences, which sometimes leads to counter-intuitive validation results. The stable model semantics does not suffer from (b) but still has problem (a): the intractability (specifically, NP-hardness) of validation in the two semantics is caused by the possible simultaneous presence of recursion and negation in constraints. NP-hardness already applies when the size of input constraints is assumed to be fixed by a constant (i.e., in *data complexity*), which becomes a major challenge for implementing validators for recursive SHACL as RDF graphs in practice are often very big. Interestingly, in the case of the supported model semantics, NP-hardness holds already for constraints with *stratified* negation. In this paper, we argue that recursive SHACL can be naturally equipped with a semantics inspired in the *well-founded semantics* for logic programs, a semantics that also plays a major role in (recursive) DATALOG queries with default negation. This semantics is not only intuitive (e.g., it avoids problem (b)), but it is also computationally tractable.

The rest of this paper is organized as follows:

- We conclude this introduction with a rather detailed motivating example to illustrate the basic ideas and the challenge addressed in this paper.
- In Section 2, we recall the well-founded semantics for propositional logic programs based on the notion of *unfounded sets*, and provide an example on how the (unique) well-founded model of a program can be computed.
- In Section 3, we recall the (abstract) syntax of SHACL and recall the (2-valued) supported model semantics of SHACL. This most basic semantics is illustrated with an example that shows an undesired self-justification cycle.
- In Section 4, we present our well-founded semantics for SHACL, based on a notion of unfounded sets (adapted from the notion used for logic programs).
- In Section 5, we describe a transformation that allows to reduce the validation problem to the task of computing the well-founded model of a propositional logic program. This transformation can be implemented efficiently and leads to a way of performing SHACL validation under the new semantics using some of the existing deductive database engines.
- In Section 6, we conclude this paper, including a discussion of some of the challenges and future directions in the research on SHACL.

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



$$ProfessorShape \leftarrow EmployeeShape \wedge (\exists \mathsf{isMentorOf}.StudentShape) \qquad (P)$$
$$EmployeeShape \leftarrow \mathsf{Employee} \vee \exists \mathsf{hasSuperior}.EmployeeShape \qquad (E)$$
$$StudentShape \leftarrow \mathsf{Student} \vee (EmployeeShape \wedge \neg ProfessorShape) \qquad (S)$$

Figure 1: Example knowledge graph and SHACL constraints

Example 1. *Consider the example knowledge graph (KG) in Figure 1, containing some information about members of an educational institution. The KG mentions four persons called* Alex, Blake, Cameron *and* Drew*, which correspond to the four nodes of the KG.* Alex *is (explicitly) stated to be an employee in this institution, which is indicated by their membership in the class* Employee. Cameron *is a (regular) student in this institution, which is indicated by their membership in the class* Student. Blake *is a mentor of* Cameron*, which is indicated via the edge between them labeled with the property* isMentorOf. Blake *has* Alex *as a superior in this institution, captured via an edge labeled with* hasSuperior. *Furthermore,* Cameron *has* Drew *as another mentor. Finally, the KG has two facts that are quite questionable:* Alex *is a mentor of* Alex*, and* Drew *is a superior of* Drew. *The presence of these two facts is clearly unintended (and should be dealt with using an appropriate validation machinery).*

The KG presented so far does not carry any semantic information on the expected relations between the different kinds of objects that are present, it contains no schema information nor integrity constraints. The SHACL standard specifies a language for expressing such information. Consider the three expressions (P), (E) *and* (S) *in Figure 1, which intuitively correspond to three* validation rules *that allow to identify nodes of three types: nodes corresponding to professors, employees, and students, respectively. Each of these three types has a name (i.e.,* $ProfessorShape$*,* $EmployeeShape$*, and* $StudentShape$*) that is associated to its* definition *using a (possibly quite complex) logic-based expression. In the DATALOG terminology, we have* $ProfessorShape$*,* $EmployeeShape$*, and* $StudentShape$ *as* monadic intensional *predicates, while* Employee, Student *are* unary extensional *predicates, and* isMentorOf, hasSuperior *are* binary extensional *predicates. Specifically,* (P) *tells us that a person is considered to be a professor (has type* $ProfessorShape$*) if they are an employee (of type* $EmployeeShape$*) and they mentor at least one student (an object with type* $StudentShape$*). According to the rule* (E)*, a person is an employee (has type* $EmployeeShape$*) if that is stated explicitly (via membership in the class* Employee*), or the person has a superior that is in turn of type* $EmployeeShape$*. The last rule* (S) *tells us that an object is of type* $StudentShape$ *if the object is explicitly stated to be an instance of the class* Student*, or the object is of type* $EmployeeShape$ *but not of type* $ProfessorShape$*. Intuitively, in our example, the employees of the institution that are not professors can be seen as students who can, e.g., take courses.*

As a final step, we need to specify which nodes of the graph are expected to satisfy selected types.

This is done via a specification of target nodes. *In the basic case, we can list pairs of nodes and the expected type (SHAPE) names. For example, consider the following target specification:*

$$\mathcal{T}_1 = \{(\text{Blake}, \mathit{ProfessorShape}), (\text{Cameron}, \mathit{StudentShape})\}$$

Intuitively, our KG equipped with the validation rules (P), (E) *and* (S) validates *the targets in* $\mathcal{T}_1$. *Indeed,* Cameron *satisfies* $\mathit{StudentShape}$ *because they are explicitly stated as an instance of* Student. *Since* Blake *has* Alex *as a superior and* Alex *is explicitly stated to be an employee, we have that* Blake *satisfies* $\mathit{EmployeeShape}$. *Given that* Blake *mentors* Cameron, *we infer that* Blake *satisfies* $\mathit{ProfessorShape}$. *Unsurprisingly, we can have targets that we don't expect to be validated by our KG and the above validation rules (under a reasonable validation semantics): this holds, e.g., for* $\mathcal{T} = \{(\text{Blake}, \mathit{StudentShape})\}$ *and* $\mathcal{T} = \{(\text{Cameron}, \mathit{ProfessorShape})\}$.

In practice, we want more succinct representations of target specifications. The SHACL standard describes the so-called class *and* property (domain and range) targets. *E.g., with* $\mathcal{T} = \{(\text{Student}, \mathit{EmployeeShape})\}$ *we ask each instance of* Student *to satisfy* $\mathit{EmployeeShape}$ *(this requirement will clearly be violated in our KG).*

Let's consider the 2-valued supported model semantics of Corman et at. [1]. Here we simply try to assign to each node some set of shape names such that: (i) a node e *is a assigned a shape name* S *iff* e *satisfies the shape expression associated to* S*, and (ii) the target specification is respected. Unfortunately, under this semantics, the target* $\mathcal{T}_1$ *above cannot be validated by our KG, which is certainly not desirable. The problem here is that* Alex *rules out the existence of any shape assignment as required for validation. If we assign* $\mathit{ProfessorShape}$ *to* Alex, *then* Alex *must be a mentor of some object with* $\mathit{StudentShape}$. *Since* Alex *only mentors themselves, and* $\mathit{StudentShape}$ *is incompatible with* $\mathit{ProfessorShape}$, *our candidate shape assignment fails. If we decide to not assign* $\mathit{ProfessorShape}$ *to* Alex, *then* Alex *must not mentor any object with* $\mathit{StudentShape}$. *However,* Alex *mentors themselves and obviously satisfies* $\mathit{StudentShape}$. *Thus again the shape assignment has failed. Overall, the 2-valued supported model semantics has a significant disadvantage: If we have some issue in the graph that bars the existence of a total shape assignment, then the validation fails for all target specifications, even those without any connection to the problematic part of the knowledge graph. The 2-valued stable model semantics in [2] has the same problem, and thus [1, 2] also consider 3-valued versions of their semantics.*

Consider the knowledge graph obtained by dropping the node Alex, *i.e., deleting the class membership and property assertions that involve* Alex. *Then a total shape assignment becomes possible: simply assign* $\mathit{StudentShape}$ *to* Cameron *and assign* $\mathit{ProfessorShape}$ *and* $\mathit{EmployeeShape}$ *to* Drew. *In fact, this is the only assignment possible. However, this situation is questionable because the knowledge graph states that* Drew *supervises themselves, which in turn causes an unfounded inference:* Drew *is an employee because* Drew *is an employee. Thus this graph validates the target specification* $\mathcal{T} = \{(\text{Drew}, \mathit{ProfessorShape})\}$ *despite the absence of evidence that* Drew *is an employee in the organization. The two problems that were discussed in this example are addressed by the well-founded semantics for SHACL that we discuss here.*

2. Preliminaries

We now recall the well-founded semantics for propositional logic programs with negation [4]. We assume a countably infinite set $\mathrm{N_{pa}}$ of *propositional atoms*, also called *positive literals*. A *negative literal* is an expression of the form $\neg a$, where $a \in \mathrm{N_{pa}}$. A *literal* is either a positive or a negative literal. A *rule* ρ is an expression of the form $h \leftarrow L_1, \ldots, L_n$ where h is a propositional atom, and $L_1, \ldots, L_n$ are literals. The atom h is called the *head* of ρ, denoted $\mathsf{head}(\rho)$. We let $\mathsf{body}(\rho) = \{L_1, \ldots, L_n\}$. A *program* P is any finite set of rules.

A (3-valued) *interpretation* I is any set of literals such that $\{a, \neg a\} \not\subseteq I$ for any a. Intuitively, if a is a propositional atom such that $a \in I$, then a is *true* in I. If $\neg a \in I$, then a is *false* in I. If neither $a \in I$ nor $\neg a \in I$, then the truth value of a is *undefined* in I.

A set U of propositional atoms is called an *unfounded set* w.r.t. an interpretation I and a program P, if at least one of the following holds for all rules $\rho \in P$:

(i) $\mathsf{head}(\rho) \notin U$,
(ii) there is a positive literal $b \in \mathsf{body}(\rho)$ such that $b \in U$ or $\neg b \in I$, or
(iii) there is a negative literal $\neg b \in \mathsf{body}(\rho)$ such that $b \in I$.

For any program P and interpretation I, there is a unique $\subseteq$-maximal unfounded set w.r.t. P and I, which we denote $U_P(I)$. For a program P, we define a pair T_P and W_P of operators that map interpretations to interpretations as follows:

$$T_P(I) = \{\mathsf{head}(\rho) \mid \mathsf{body}(\rho) \subseteq I, \rho \in P\}$$
$$W_P(I) = T_P(I) \cup \{\neg a \mid a \in U_P(I)\}$$

Intuitively, by applying the operator W_P on an interpretation I, we augment I with some new literals: (i) first we include $T_P(I)$ that contains immediate consequences of the rules in P and the literals in I, and then (ii) we add the negation of all atoms that can be safely assumed false according to $U_P(I)$. The operator W_P is monotonic, i.e., $W_P(I_1) \subseteq W_P(I_2)$ holds whenever $I_1 \subseteq I_2$. Thus W_P has the least fixpoint *lfp*(W_P) which we call the *well-founded model of* P, and denote it by $WFS(P)$. In more detail, we can compute $WFS(P)$ by considering the sequence $I_0, I_1, \ldots$ with $I_0 = \emptyset$ and $I_i = W_P(I_{i-1})$ for $i > 0$. Due to the monotonicity of W_P, this sequence reaches a j such that $I_{j+1} = I_j$. Then $WFS(P) = I_j$.

Example 2. *Consider the program P consisting of the following rules:*

$$a \leftarrow b, c \qquad b \leftarrow a, d \qquad d \leftarrow e \qquad c \leftarrow \neg b \qquad e \leftarrow \neg d$$

Let us compute $WFS(P)$. We start with $I_0 = \emptyset$. We have $T_P(I_0) = \emptyset$ and $U_P(I_0) = \{a, b\}$. Thus $I_1 = W_P(I_0) = \{\neg a, \neg b\}$. To compute I_2, first note that $T_P(I_1) = \{c\}$ and $U_P(I_1) = \{a, b\}$. Then $I_2 = W_P(I_1) = \{c, \neg a, \neg b\}$. Observe that $U_P(I_2) = \{a, b\}$ and $T_P(I_2) = \{c\}$, and thus $I_3 = I_2$. Hence $WFS(P) = \{c, \neg a, \neg b\}$.

3. SHACL Constraints

We recall the (abstract) syntax of SHACL constraints, mainly following the approach in [1]. We assume countably infinite, mutually disjoint sets $\mathrm{N}_{\mathrm{node}}, \mathrm{N}_{\mathrm{class}}, \mathrm{N}_{\mathrm{prop}}, \mathrm{N}_{\mathrm{shape}}, \mathrm{N}_{\mathrm{var}}$ of *nodes*, *class names*, *property names*, *shape names*, and *variables*, respectively. If $p \in \mathrm{N}_{\mathrm{prop}}$, then p^- is the *inverse property* of p. We let $\mathrm{N}^{\pm}_{\mathrm{prop}} = \mathrm{N}_{\mathrm{prop}} \cup \{p^- \mid p \in \mathrm{N}_{\mathrm{prop}}\}$ and let $(p^-)^- = p$ for any $p \in \mathrm{N}_{\mathrm{prop}}$. Each $r \in \mathrm{N}^{\pm}_{\mathrm{prop}}$ is a *basic property*. A *property path* is an expression E built using the grammar

$$E ::= r \mid E \circ E \mid E \cup E \mid E^*$$

where $r \in \mathrm{N}^{\pm}_{\mathrm{prop}}$.

A *data graph* is a pair $\mathcal{G} = (\Delta^{\mathcal{G}}, \cdot^{\mathcal{G}})$, where $\Delta^{\mathcal{G}} \subseteq \mathrm{N}_{\mathrm{node}}$ is a set of nodes (called *domain*), and $\cdot^{\mathcal{G}}$ is a function that assigns to every class name $A \in \mathrm{N}_{\mathrm{class}}$ some set of nodes $A^{\mathcal{G}} \subseteq \Delta^{\mathcal{G}}$, and to every property name $p \in \mathrm{N}_{\mathrm{prop}}$ some binary relation $p^{\mathcal{G}} \subseteq \Delta^{\mathcal{G}} \times \Delta^{\mathcal{G}}$. For a property name p, we let $(p^-)^{\mathcal{G}} = \{(o', o) \mid (o, o') \in p^{\mathcal{G}}\}$. Furthermore, the function $\cdot^{\mathcal{G}}$ is extended to all property paths as follows:

$$(E_1 \circ E_2)^{\mathcal{G}} = (E_1)^{\mathcal{G}} \circ (E_2)^{\mathcal{G}}$$
$$(E_1 \cup E_2)^{\mathcal{G}} = (E_1)^{\mathcal{G}} \cup (E_2)^{\mathcal{G}}$$
$$(E^*)^{\mathcal{G}} = (E^{\mathcal{G}})^*$$

SHACL constraints. A *shape expression* φ is an expression built according to the following grammar:

$$\varphi ::= c \mid A \mid \neg A \mid s \mid \neg s \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid X \mid \exists E.\varphi$$

where $c \in \mathrm{N}_{\mathrm{node}}$, $A \in \mathrm{N}_{\mathrm{class}}$, $s \in \mathrm{N}_{\mathrm{shape}}$, $X \in \mathrm{N}_{\mathrm{var}}$, and E is a complex path. A *shape constraint* is an expression of the form $s \leftarrow \varphi$, where $s \in \mathrm{N}_{\mathrm{shape}}$ and φ is a shape expression. In the following, when considering a set $\mathcal{C}$ of shape constraints, we always require that (a) $\varphi_1 = \varphi_2$ for any pair $s \leftarrow \varphi_1$ and $s \leftarrow \varphi_2$ of shape constraints in $\mathcal{C}$, and (b) for every shape name s that appears in $\mathcal{C}$, there is φ such that $s \leftarrow \varphi \in \mathcal{C}$. The above just says that every shape name that appears in C has exactly one "definition". Concretely, for such a constraint set $\mathcal{C}$ and a shape name s that appears in $\mathcal{C}$, we let $\mathcal{C}(s)$ denote the unique φ such that $s \leftarrow \varphi \in C$. A shape expression is *ground* if it contains no variables. In what follows we only consider ground shape expressions, with a small exception in Section 5 where variables are used as an auxiliary tool to obtain a translation into logic programs. Note that the SHACL standard does not allow variables in shape expressions. We also note that the syntax above does not support *property pair equality*, *property pair disjointness*, *cardinality constraints*, and negation of complex expressions; these features are described in the W3C standard, but we omit them here for presentation reasons (they will be added and discussed in the extended version of this paper).

A *target* is a pair (ℓ, s), where $\ell \in \mathrm{N}_{\mathrm{node}} \cup \mathrm{N}_{\mathrm{class}} \cup \mathrm{N}^{\pm}_{\mathrm{prop}}$ and $s \in \mathrm{N}_{\mathrm{shape}}$. A *SHACL document* is a pair $(\mathcal{C}, \mathcal{T})$, where $\mathcal{C}$ is a set of shape constraints and $\mathcal{T}$ is a set of targets. We will next discuss the *validation* of a graph $\mathcal{G}$ against a SHACL document $(\mathcal{C}, \mathcal{T})$. For this we require that the nodes that explicitly appear in $\mathcal{C}$ or $\mathcal{T}$ are also included in the domain $\Delta^{\mathcal{G}}$.

$$
\begin{aligned}
c^{\mathcal{G},\mu} &= \{c\} \\
s^{\mathcal{G},\mu} &= \mu(s) \\
A^{\mathcal{G},\mu} &= A^{\mathcal{G}} \\
(\neg A)^{\mathcal{G},\mu} &= \Delta^{\mathcal{G}} \setminus A^{\mathcal{G}} \\
(\neg s)^{\mathcal{G},\mu} &= \Delta^{\mathcal{G}} \setminus \mu(s) \\
(\varphi_1 \wedge \varphi_2)^{\mathcal{G},\mu} &= (\varphi_1)^{\mathcal{G},\mu} \cap (\varphi_2)^{\mathcal{G},\mu} \\
(\varphi_1 \vee \varphi_2)^{\mathcal{G},\mu} &= (\varphi_1)^{\mathcal{G},\mu} \cup (\varphi_2)^{\mathcal{G},\mu} \\
(\exists E.\varphi)^{\mathcal{G},\mu} &= \{e \in \Delta^{\mathcal{G}} \mid \exists e' : (e,e') \in E^{\mathcal{G}} \wedge e' \in \varphi^{\mathcal{G},\mu}\}
\end{aligned}
$$

Figure 2: Evaluating complex shape expressions

Supported model semantics. We present next the basic 2-valued semantics of recursive SHACL due to Corman et al. [1]. A *shape assignment* for $\mathcal{G}$ is any function μ that maps every shape name $s \in \mathrm{N}_{\mathrm{shape}}$ to a subset $\mu(s) \subseteq \Delta^{\mathcal{G}}$. For a data graph $\mathcal{G}$ and a shape assignment μ for $\mathcal{G}$, we define a function $\cdot^{\mathcal{G},\mu}$ that maps every shape expression φ to a set $\varphi^{\mathcal{G},\mu}$ of nodes as shown in Figure 2. We say a graph $\mathcal{G}$ is *valid* against a SHACL document $(\mathcal{C}, \mathcal{T})$, if there exists a shape assignment μ such that:

1. $\mu(s) = \varphi^{\mathcal{G},\mu}$ for all constraints $s \leftarrow \varphi \in \mathcal{C}$, and
2. for all $(\ell, s) \in T$ we have:
   - if ℓ is a node $a \in \mathrm{N}_{\mathrm{node}}$, then $a \in \mu(s)$,
   - if ℓ is a class name A, then $A^{\mathcal{G}} \subseteq \mu(s)$,
   - if ℓ is a property name p, then $\{e \mid (e,e') \in p^{\mathcal{G}}\} \subseteq \mu(s)$, and
   - if $\ell = p^{-}$ for a property name p, then $\{e' \mid (e,e') \in p^{\mathcal{G}}\} \subseteq \mu(s)$.

Example 3. *Consider a SHACL document $(\mathcal{C}, \mathcal{T})$ consisting of the following:*

$$\mathcal{C} = \{s_1 \leftarrow \exists p.\neg A, \quad s_2 \leftarrow \neg s_1, \quad s_3 \leftarrow B \wedge \neg s_4, \quad s_4 \leftarrow B \wedge \neg s_3, \quad s_5 \leftarrow \exists r.s_5\}$$

$$\mathcal{T} = \{(A, s_1), (B, s_2), (a, s_5)\}$$

Consider a data graph $\mathcal{G}$ with $\Delta^{G} = \{a, b\}$ and

$$A^{\mathcal{G}} = \{a\} \quad B^{\mathcal{G}} = \{b\} \quad p^{\mathcal{G}} = \{(a,b)\} \quad r^{\mathcal{G}} = \{(a,a)\}$$

It easy to see that $\mathcal{G}$ is valid against $(\mathcal{C}, \mathcal{T})$, which is witnessed, e.g., by the shape assignment μ such that $\mu(s_1) = \mu(s_5) = \{a\}, \mu(s_2) = \mu(s_3) = \{b\}, \mu(s_4) = \emptyset$. This example already illustrates a drawback of the supported model semantics: a is inferred to satisfy s_5 because it has an r-connection to a node that satisfies s_5. However, a has only a singe r-connection to itself, i.e., we have self-justification.

Consider the targets $\mathcal{T}_1 = \{(b, s_3)\}$, $\mathcal{T}_2 = \{(b, s_4)\}$, $\mathcal{T}_3 = \{(b, s_3), (b, s_4)\}$. Observe that $\mathcal{G}$ is valid against $(\mathcal{C}, \mathcal{T}_1)$ and $(\mathcal{C}, \mathcal{T}_2)$ individually, but $\mathcal{G}$ does not validate $\mathcal{T}_3$ as we cannot assign both s_3

$$
\begin{aligned}
\lfloor s \rfloor_S^{\mathcal{G}} &= \{a \mid s(a) \in S\} \\
\lfloor \neg s \rfloor_S^{\mathcal{G}} &= \{a \mid \neg s(a) \in S\} \\
\lceil s \rceil_S^{\mathcal{G}} &= \{a \in \Delta^{\mathcal{G}} \mid \neg s(a) \notin S\} \\
\lceil \neg s \rceil_S^{\mathcal{G}} &= \{a \in \Delta^{\mathcal{G}} \mid s(a) \notin S\} \\
[c]_S^{\mathcal{G}} &= \{c\} && [\cdot] \in \{\lfloor \cdot \rfloor, \lceil \cdot \rceil\} \\
[A]_S^{\mathcal{G}} &= \{a \mid a \in A^{\mathcal{G}}\} && [\cdot] \in \{\lfloor \cdot \rfloor, \lceil \cdot \rceil\} \\
[\neg A]_S^{\mathcal{G}} &= \{a \mid a \in \Delta^{\mathcal{G}} \setminus A^{\mathcal{G}}\} && [\cdot] \in \{\lfloor \cdot \rfloor, \lceil \cdot \rceil\} \\
[\varphi_1 \wedge \varphi_2]_S^{\mathcal{G}} &= [\varphi_1]_S^{\mathcal{G}} \cap [\varphi_2]_S^{\mathcal{G}} && [\cdot] \in \{\lfloor \cdot \rfloor, \lceil \cdot \rceil\} \\
[\varphi_1 \vee \varphi_2]_S^{\mathcal{G}} &= [\varphi_1]_S^{\mathcal{G}} \cup [\varphi_2]_S^{\mathcal{G}} && [\cdot] \in \{\lfloor \cdot \rfloor, \lceil \cdot \rceil\} \\
[\exists E.\varphi]_S^{\mathcal{G}} &= \{a \mid \exists a' : (a, a') \in E^{\mathcal{G}} \wedge a' \in [\varphi]_S^{\mathcal{G}}\} && [\cdot] \in \{\lfloor \cdot \rfloor, \lceil \cdot \rceil\}
\end{aligned}
$$

Figure 3: Evaluating shape expressions: upper and lower bounds

and s_4 to b in a single shape assignment. This phenomenon could be undesired in certain situations: when a stronger notion of validation is desired, we may expect that a graph validates $\mathcal{T}' \cup \mathcal{T}''$ whenever it validates $\mathcal{T}'$ and $\mathcal{T}''$. The well-founded semantics introduced next has this feature.

Note that by introducing an inconsistency in $\mathcal{C}$, e.g., by adding the constraint $s_6 \leftarrow \neg s_6$, we lose the existence of a shape assignment, and thus no target gets validated, including $\mathcal{T} = \emptyset$. This is partially resolved by considering a 3-valued supported model semantics, which does not require determining the truth value of all shape names [1]. However, this does not resolve the problem of self-justifications, which can be dealt with by using the 2-valued and 3-valued stable model semantics of Andreşel et al. [2]. See also [5] for an approach based on Magic Sets to increase inconsistency tolerance of the 2-valued stable model semantics in SHACL.

4. Well-founded Semantics for SHACL

We are now ready to define a well-founded semantics for SHACL. We do this by adapting the notion of *unfounded sets* and fixpoint computations from [4]. We first need our version of 3-valued interpretations in the context of SHACL.

A *shape atom* is an expression of the form $s(a)$, where $s \in \mathrm{N}_{\mathrm{shape}}$ and $a \in \mathrm{N}_{\mathrm{node}}$. A *negated shape atom* is an expression $\neg s(a)$, where $s(a)$ is an atom. A *(shape) literal* is a possibly negated shape atom. A *(3-valued) interpretation* (for SHACL constraints) is any set S of shape literals such that there is no $s(a) \in S$ with $\neg s(a) \in S$. Intuitively, an atom $s(a) \in S$ means that there is a justification that s holds at a, a literal $\neg s(a) \in S$ means that there is no reason for s to hold at a, while $\{s(a), \neg s(a)\} \cap S = \emptyset$ corresponds to the case where the the satisfaction of s at a is *undefined*. For a set K of shape atoms, let $\neg.K = \{\neg s(a) \mid s(a) \in K\}$.

Definition 1. *For a given graph $\mathcal{G}$ and an interpretation S, we define two functions $\lfloor \cdot \rfloor_S^{\mathcal{G}}$ and $\lceil \cdot \rceil_S^{\mathcal{G}}$ that map every shape expression to a set of nodes. These functions are defined in Figure 3.*

Assume a data graph $\mathcal{G}$ and a constraint $s \leftarrow \varphi$. Suppose we "believe" a set S of shape literals, i.e., we assume that all literals in S are true. Then $\lfloor\varphi\rfloor_S^{\mathcal{G}}$ and $\lceil\varphi\rceil_S^{\mathcal{G}}$ return us the nodes of $\mathcal{G}$ where φ *is certainly true* and where φ *is possibly true*, respectively. Thus $\lfloor\varphi\rfloor_S^{\mathcal{G}}$ can be used to infer positive shape literals: if $a \in \lfloor\varphi\rfloor_S^{\mathcal{G}}$, then we can infer $s(a)$. We can use $\lceil\varphi\rceil_S^{\mathcal{G}}$ to infer negative information: if $a \notin \lceil\varphi\rceil_S^{\mathcal{G}}$, then we can infer $\neg s(a)$. These inferences are formalized next.

Definition 2. *Assume a data graph $\mathcal{G}$ and a set $\mathcal{C}$ of constraints. We define an operator $T_{\mathcal{G},\mathcal{C}}(\cdot)$ that maps interpretations into interpretations as follows:*

$$T_{\mathcal{G},\mathcal{C}}(S) = \{s(a) \mid s \leftarrow \varphi \in C, a \in \lfloor\varphi\rfloor_S^{\mathcal{G}}\}$$

We are now ready to define the notion of an unfounded set of shape atoms.

Definition 3 (Unfounded set). *Assume an interpretation S, a data graph $\mathcal{G}$, and a set $\mathcal{C}$ of constraints. A set U of shape atoms is called an* unfounded set *w.r.t. S, $\mathcal{G}$ and $\mathcal{C}$, if $a \notin \lceil\mathcal{C}(s)\rceil_{S\cup\neg.U}^{\mathcal{G}}$ for all $s(a) \in U$.*

Assume a data graph $\mathcal{G}$, a set U of shape atoms, and suppose we "believe" a set S of shape literals. Intuitively, the atoms in U form an unfounded set (and can thus be simultaneously set to *false*) if none of the shape atoms $s(a) \in U$ can possibly be implied by the associated constraint, assuming the negation of the atoms in U holds (in addition to S being true).

The following property follows from the fact that $U_1 \cup U_2$ is an unfounded set w.r.t. S, $\mathcal{G}$ and $\mathcal{C}$ whenever U_1, U_2 are two unfounded sets w.r.t. S, $\mathcal{G}$ and $\mathcal{C}$.

Proposition 1. *Assume an interpretation S, a data graph $\mathcal{G}$, and a set $\mathcal{C}$ of constraints. There exists a unique set U such that*

1. *U is an unfounded set w.r.t. S, $\mathcal{G}$ and $\mathcal{C}$, and*
2. *there is no $U' \supset U$ that is unfounded set w.r.t. S, $\mathcal{G}$ and $\mathcal{C}$.*

The unique set U in the proposition above is called the *greatest unfounded set* w.r.t. S, $\mathcal{G}$ and $\mathcal{C}$. Assume a data graph $\mathcal{G}$ and a set $\mathcal{C}$ of constraints. We let $U_{\mathcal{G},\mathcal{C}}$ be the operator that maps every interpretation S to the greatest unfounded set w.r.t. S, $\mathcal{G}$ and $\mathcal{C}$. Thus, $U_{\mathcal{G},\mathcal{C}}(S)$ is the maximal set of shape atoms that we can safely set to *false* if we assume that the literals in S are true. We can now finally define a well-founded semantics for SHACL. We define an operator $W_{\mathcal{G},\mathcal{C}}$ that maps interpretations into interpretations as follows: It combines the positive consequences based on the $T_{\mathcal{G},\mathcal{C}}$ operator and the negated atoms of the greatest unfounded set produced by the $U_{\mathcal{G},\mathcal{C}}$ operator. More formally, we define:

$$W_{\mathcal{G},\mathcal{C}}(S) = T_{\mathcal{G},\mathcal{C}}(S) \cup \neg.U_{\mathcal{G},\mathcal{C}}(S)$$

The above operator is monotone, i.e., $W_{\mathcal{G},\mathcal{C}}(S_1) \subseteq W_{\mathcal{G},\mathcal{C}}(S_2)$ whenever S_1, S_2 are two interpretations with $S_1 \subseteq S_2$. Thus $W_{\mathcal{G},\mathcal{C}}$ has the least fixpoint, i.e., there exists S such that (i) $W_{\mathcal{G},\mathcal{C}}(S) = S$, and (ii) there is no $S' \subset S$ with $W_{\mathcal{G},\mathcal{C}}(S') = S'$. We use $WFS(\mathcal{G},\mathcal{C})$ to denote the least fixpoint of $W_{\mathcal{G},\mathcal{C}}$, and call it the *well-founded model* of $\mathcal{G}$ and $\mathcal{C}$. $WFS(\mathcal{G},\mathcal{C})$ can be obtained by constructing a sequence $S_0, S_1, S_2, \ldots$ such that $S_0 = \emptyset$ and $S_{i+1} = W_{\mathcal{G},\mathcal{C}}(S_i)$ until eventually an index j is reached were $S_j = S_{j+1}$. Then $WFS(\mathcal{G},\mathcal{C}) = S_j$.

We can now define validation under the well-founded semantics. We say a graph $\mathcal{G}$ is *valid* against a SHACL document $(\mathcal{C}, \mathcal{T})$ (under the well-founded semantics), if for all $(\ell, s) \in T$ we have:

- if ℓ is a node $a \in \mathrm{N}_{\mathrm{node}}$, then $s(a) \in \mathit{WFS}(\mathcal{G}, \mathcal{C})$,
- if ℓ is a class name A, then $s(a) \in \mathit{WFS}(\mathcal{G}, \mathcal{C})$ for all $a \in A^{\mathcal{G}}$,
- if ℓ is a property name p, then $s(a) \in \mathit{WFS}(\mathcal{G}, \mathcal{C})$ for all a with $(a, b) \in p^{\mathcal{G}}$,
- if $\ell = p^{-}$ for $p \in \mathrm{N}_{\mathrm{prop}}$, then $s(b) \in \mathit{WFS}(\mathcal{G}, \mathcal{C})$ for all b with $(a, b) \in p^{\mathcal{G}}$.

Example 4. *Consider the data graph $\mathcal{G}$ and the constraint set $\mathcal{C}$ from Example 1. It is not difficult to check that $\mathit{WFS}(\mathcal{G}, \mathcal{C})$ contains $\mathit{ProfessorShape}(\mathit{Blake})$, $\mathit{StudentShape}(\mathit{Cameron})$, $\mathit{EmployeeShape}(\mathit{Alex})$, $\mathit{EmployeeShape}(\mathit{Blake})$. Observe also that $\mathit{ProfessorShape}(\mathit{Alex}) \notin \mathit{WFS}(\mathcal{G}, \mathcal{C})$ and $\neg\mathit{ProfessorShape}(\mathit{Alex}) \notin \mathit{WFS}(\mathcal{G}, \mathcal{C})$, i.e., membership of Alex in $\mathit{ProfessorShape}$ is undetermined, which is intuitive as discussed in Example 1. For the remaining shape atoms over the signature of $\mathcal{G}$ and $\mathcal{C}$, we have that $\mathit{WFS}(\mathcal{G}, \mathcal{C})$ contains their negation. Thus $\mathcal{G}$ is valid against $(\mathcal{C}, \{(\mathit{Blake}, \mathit{ProfessorShape})\})$ but not valid w.r.t. $(\mathcal{C}, \{(\mathit{Drew}, \mathit{ProfessorShape})\})$.*

5. From SHACL to Propositional Logic Programs

Assume a graph $\mathcal{G}$ and a set $\mathcal{C}$ of constraints. We show how to compute a propositional logic program $P_{\mathcal{G},\mathcal{C}}$ from $\mathcal{G}$ and $\mathcal{C}$ such that the well-founded model of $P_{\mathcal{G},\mathcal{C}}$ corresponds to the well-founded model of $\mathcal{G}$ and $\mathcal{C}$. We define our target program as $P_{\mathcal{G},\mathcal{C}} = \bigcup_{s \leftarrow \varphi \in \mathcal{C}} P_{s,\varphi}$, where each $P_{s,\varphi}$ is defined as follows.

Assume a constraint $s \leftarrow \varphi$. First, perform the following replacement in φ exhaustively until no further replacement is possible: find a position in φ where a negative shape literal $\neg s$ occurs, and replace $\neg s$ by a fresh variable $x_{\neg s}$. Second, perform the following replacement in φ exhaustively: find a position in φ where a positive shape literal s occurs, and replace s by a fresh variable x_s. After these steps, we have that the shape expression φ' that is produced from φ has a tuple of variables $\vec{x} = (x^1_{\neg s_1}, \ldots, x^k_{\neg s_k}, x^{k+1}_{s_{k+1}}, \ldots, x^n_{s_n})$. Then the program $P_{s,\varphi}$ consists of the propositional rule $s(a) \leftarrow \neg s_1(b_1), \ldots, \neg s_k(b_k), s_{k+1}(b_{k+1}), \ldots, s_n(b_n)$ for every node a and every node tuple $\vec{b} = (b_1, \ldots, b_n)$ such that $a \in (\varphi'[\vec{x}/\vec{b}])^{\mathcal{G},\emptyset}$. Here $\varphi[\vec{x}/\vec{b}]$ denotes the shape expression that is obtained from φ by performing the substitution of variables $\vec{x}$ by nodes $\vec{b}$.

Proposition 2. *If $\mathcal{G}$ is a graph and $\mathcal{C}$ is a constraint set, then $\mathit{WFS}(\mathcal{G}, \mathcal{C}) = \mathit{WFS}(P_{\mathcal{G},\mathcal{C}})$. The size of $P_{\mathcal{G},\mathcal{C}}$ is exponential in the size of $\mathcal{G}$ and $\mathcal{C}$, but it is only polynomial in case the number of shape names in every constraint is bounded by a constant.*

Note that, for a given $s \leftarrow \varphi$, the relevant node tuples $(a, b_1, \ldots, b_n)$ above can be computed by posing a single SPARQL query over $\mathcal{G}$, in a similar way as in [6]. This opens a way to implement validation under the new semantics by exploiting existing SPARQL endpoints. We also remark that, by introducing fresh shape names, we can rewrite a given SHACL document in a way that its shape expressions contain at most 2 shape names, thus obtaining a polynomial transformation from SHACL to propositional logic programs under the well-founded semantics. However, our initial experiments show that aiming for a guaranteed polynomial time translation does not necessarily lead to increased efficiency of validation.

6. Discussion

In this paper we have defined a well-founded semantics for SHACL constraints with cyclic dependencies. This semantics is not only intuitive and based on well-established principles in logic programming and deductive databases, but it also has advantages over the previous approaches (based on supported and stable models) in terms of semantic and computational properties, i.e., it avoids unfounded justifications and is tractable. We note that our semantics has a direct connection to the so-called *cautious validation* under the 3-valued stable model semantics in [2], which is due to the classic result by Przymusinski [7].

We have also established a connection between the SHACL semantics introduced here and the well-founded semantics for logic programs. We are currently working on an implementation of a SHACL validator that directly exploits this connection: Given a graph $\mathcal{G}$ and a constraint set $\mathcal{C}$, it issues SPARQL queries over $\mathcal{G}$ in order to compute a propositional logic program $P_{\mathcal{G},\mathcal{C}}$, which can then be sent to a dedicated reasoner to compute the well-founded model of $P_{\mathcal{G},\mathcal{C}}$. Our initial experiments using the DLV engine [8] suggest that this is a viable approach; these results will be presented in the extended version of this paper.

There are many research questions for future work. For instance, the computational complexity of satisfiability and implication of SHACL constraints under the well-founded semantics is a natural next topic for investigations. Some initial results in this area for the semantics introduced previously were presented in [9, 10]. Due to the syntactic form of shape expressions, many of those previous results are related to known positive and negative decidability and complexity results in the area of Description Logics. More broadly, SHACL is related to the problem of reconciling the open-world assumption (as in OWL and Description Logics) with the closed-world assumption (as in databases and logic programming). E.g., validation in SHACL under the supported model semantics can be seen as checking the satisfiability of a Description Logic ontology where all but some selected concept names are *closed predicates*. Supporting ontological inferences in the context of SHACL validation is an interesting research direction, which is also hinted at by the SHACL standard. More details on the challenges of combining open- and closed-world assumptions can be found in [11, 12]. Another natural direction is to study *explanations* and *repairs* of violations of SHACL constraints under the well-founded semantics: these topics are natural as the SHACL standard calls for (but does not specify the details of) the so-called *validation reports* to support users and applications. Some initial work on explaining and repairing violations of SHACL constraints was reported in [13, 14].

Acknowledgments

This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation. It was also partially supported by the Austrian Science Fund (FWF) projects P30360 and P30873, and by the Vienna Business Agency's project CoRec.

References

[1] J. Corman, J. L. Reutter, O. Savkovic, Semantics and validation of recursive SHACL, in: Proc. of ISWC 2018, volume 11136 of *LNCS*, Springer, 2018, pp. 318–336. URL: https://doi.org/10.1007/978-3-030-00671-6_19.

[2] M. Andreşel, J. Corman, M. Ortiz, J. L. Reutter, O. Savkovic, M. Šimkus, Stable model semantics for recursive SHACL, in: WWW '20: The Web Conference 2020, ACM / IW3C2, 2020, pp. 1570–1580. URL: https://doi.org/10.1145/3366423.3380229.

[3] P. Pareti, G. Konstantinidis, A review of SHACL: from data validation to schema reasoning for RDF graphs, in: [15], 2021. URL: https://doi.org/10.1007/978-3-030-95481-9_6.

[4] A. Van Gelder, K. A. Ross, J. S. Schlipf, The well-founded semantics for general logic programs, J. ACM 38 (1991) 619–649. URL: https://doi.org/10.1145/116825.116838.

[5] S. Ahmetaj, B. Loehnert, M. Ortiz, M. Šimkus, Magic shapes for validation in SHACL, Proceedings of the VLDB Endowment. Volume 15 Issue 10 (to appear) (2022).

[6] J. Corman, F. Florenzano, J. L. Reutter, O. Savkovic, Validating SHACL constraints over a SPARQL endpoint, in: Proc. of ISWC 2019, volume 11778 of *LNCS*, Springer, 2019, pp. 145–163. URL: https://doi.org/10.1007/978-3-030-30793-6_9.

[7] T. C. Przymusinski, The well-founded semantics coincides with the three-valued stable semantics, Fundam. Inform. 13 (1990) 445–463.

[8] N. Leone, G. Pfeifer, W. Faber, T. Eiter, G. Gottlob, S. Perri, F. Scarcello, The dlv system for knowledge representation and reasoning., ACM Trans. Comput. Log. 7 (2006) 499–562. URL: http://dblp.uni-trier.de/db/journals/tocl/tocl7.html#LeonePFEGPS06.

[9] P. Pareti, G. Konstantinidis, F. Mogavero, Satisfiability and containment of recursive SHACL, Journal of Web Semantics 74 (2022) 100721. URL: https://www.sciencedirect.com/science/article/pii/S1570826822000130. doi:https://doi.org/10.1016/j.websem.2022.100721.

[10] M. Leinberger, P. Seifer, T. Rienstra, R. Lämmel, S. Staab, Deciding SHACL shape containment through description logics reasoning, in: Proc. of ISWC 2020, Springer, 2020.

[11] M. Knorr, On combining ontologies and rules, in: [15], 2021, pp. 22–58. URL: https://doi.org/10.1007/978-3-030-95481-9_2.

[12] T. Schneider, M. Šimkus, Ontologies and data management: A brief survey, Künstliche Intell. 34 (2020) 329–353. URL: https://doi.org/10.1007/s13218-020-00686-3.

[13] S. Ahmetaj, R. David, M. Ortiz, A. Polleres, B. Shehu, M. Šimkus, Reasoning about Explanations for Non-validation in SHACL, in: Proc. of KR 2021, 2021, pp. 12–21. URL: https://doi.org/10.24963/kr.2021/2.

[14] R. David, S. Ahmetaj, A. Polleres, M. Šimkus, Repairing SHACL constraint violations using answer set programming, in: Proc. of ISWC 2022 (to appear), Springer, 2022.

[15] M. Šimkus, I. Varzinczak (Eds.), Reasoning Web. Declarative Artificial Intelligence - 17th International Summer School 2021, 2021, Tutorial Lectures, volume 13100 of *LNCS*, Springer, 2022. URL: https://doi.org/10.1007/978-3-030-95481-9.