


Expressiveness of SHACL Features

Bart Bogaerts 

Vrije Universiteit Brussel, Belgium

Maxime Jakubowski 

Hasselt University, Diepenbeek, Belgium

Jan Van den Bussche 

Hasselt University, Diepenbeek, Belgium

Abstract

SHACL is a W3C-proposed schema language for expressing structural constraints on RDF graphs. Recent work on formalizing this language has revealed a striking relationship to description logics. SHACL expressions can use four fundamental features that are not so common in description logics. These features are zero-or-one path expressions; equality tests; disjointness tests; and closure constraints. Moreover, SHACL is peculiar in allowing only a restricted form of expressions (so-called targets) on the left-hand side of inclusion constraints.

The goal of this paper is to obtain a clear picture of the impact and expressiveness of these features and restrictions. We show that each of the four features is primitive: using the feature, one can express boolean queries that are not expressible without using the feature. We also show that the restriction that SHACL imposes on allowed targets is inessential, as long as closure constraints are not used.

2012 ACM Subject Classification Information systems → Semantic web description languages; Information systems → Query languages; Theory of computation → Description logics; Theory of computation → Finite Model Theory

Keywords and phrases Expressive power, schema languages

Digital Object Identifier 10.4230/LIPIcs.ICDT.2022.15

Funding Supported by AI Research Flanders.

Acknowledgements We thank Dörthe Arndt for inspiring discussions.

1 Introduction

On the Web, the Resource Description Framework (RDF [16]) is a standard format for representing knowledge and publishing data. RDF represents information in the form of directed graphs, where labeled edges indicate properties of nodes. To facilitate more effective access and exchange, it is important for a consumer of an RDF graph to know what properties to expect, or, more generally, to be able to rely on certain structural constraints that the graph is guaranteed to satisfy. We therefore need a declarative language in which such constraints can be expressed formally. In database terms, we need a schema language.

Two prominent proposals in this vein have been ShEx [6] and SHACL [18]. Both approaches use formulas that express the presence or absence of certain properties of a node or its neighbors in the graph. Such formulas are called “shapes.” When we evaluate a shape on a node, that node is called the “focus node.” Some examples of shapes, expressed for now in English, could be the following:¹

¹ In real RDF, names of properties and nodes must conform to IRI syntax, but in this paper, to avoid clutter, we take the liberty to use simple names.



15:2 Expressiveness of SHACL Features

1. “The focus node has a `phone` property, but no `email`.”
2. “The focus node has at least five incoming `managed-by` edges.”
3. “Through a path of `friend` edges, the focus node can reach a node with a `CEO-of` edge to the node `Apple`.”
4. “The focus node has at least one `colleague` who is also a `friend`.”
5. “The focus node has no other properties than `name`, `address`, or `birthdate`.”

In this paper, we look deeper into SHACL, the language recommended by the World Wide Web Consortium. We do not use the actual SHACL syntax, but work with the elegant formalization proposed by Corman, Reutter and Savkovic [9], and used in subsequent works by several authors [2, 13, 15]. That formalization reveals a striking similarity between shapes on the one hand, and concepts, familiar from description logics [4], on the other hand. The similarity between SHACL and description logics runs even deeper when we account for targeting, which is the actual mechanism to express constraints on an RDF graph using shapes.

Specifically, a non-recursive *shape schema*² is essentially a finite list of shapes, where each shape ϕ is additionally associated with a target query q . An RDF graph G is said to conform to such a schema if for every target–shape combination (q, ϕ) , and every node v returned by q on G , we have that v satisfies ϕ in G . Let us see some examples of target–shape pairs, still expressed in English:

6. “Every node of type `Person` has an `email` or `phone` property.” Here, the target query returns all nodes with an edge labeled `type` to node `Person`; the shape checks that the focus node has an `email` or `phone` property.
7. “Different nodes never have the same `email`.” Here the target query returns all nodes with an incoming `email` edge, and the shape checks that the focus node does not have two or more incoming `email` edges.
8. “Every mathematician has a finite Erdős number.” Here the target query returns all nodes of type `Mathematician`, and the shape checks that the focus node can reach the node `Erdős` by a path that matches the regular expression $(\text{author}^-/\text{author})^*$. Here, the minus superscript denotes an inverse edge.

Now interestingly, and apparent in the examples 6–8, the target queries considered for this purpose in SHACL, as well as in ShEx, actually correspond to simple cases of shapes. It is then only a small step to consider *generalized* shape schemas as finite sets of inclusion statements of the form $\phi_1 \subseteq \phi_2$, where ϕ_1 and ϕ_2 are shapes. Since, as noted above, shapes correspond to concepts, we thus see that shape schemas correspond to TBoxes in description logics.

We stress that the task we are focusing on in this paper is checking conformance of RDF graphs against shape schemas. Every shape schema \mathcal{S} defines a decision problem: given an RDF graph G , check whether G conforms to \mathcal{S} . In database terms, we are processing a *boolean query* on a graph database. In description logic terms, this amounts to *model checking* of a TBox: given an interpretation, check whether it satisfies the TBox. Thus our focus is a bit different from that of typical applications of description logics. There, facts are declared in ABoxes, which should not be confused with interpretations. The focus is then on higher reasoning tasks, such as checking satisfiability of an ABox and a TBox, or deciding logical entailment.

² Real SHACL uses the term *shapes graph* instead of shape schema.

Given the above context, let us now look in more detail at the logical constructs that can be used to build shapes. Some of these constructs are well known concept constructors from expressive description logics [7]: the boolean connectives; constants; qualified number restriction (a combination of existential quantification and counting); and regular path expressions with inverse. To illustrate, example shapes 1–3 are expressible as follows:

1. $\geq_1 \text{phone}.\top \wedge \neg \geq_1 \text{email}.\top$. This uses qualified number restriction with count 1 (so essentially existential quantification), conjunction, and negation; \top stands for true.
2. $\geq_5 \text{managed-by}^-.\top$. This uses counting to 5, and inverse.
3. $\geq_1 \text{friend}^*/\text{CEO-of}.\{\text{Apple}\}$. This uses a regular path expression and the constant `Apple`.

However, SHACL also has four specific logical features that are less common:

Equality, disjointness: The shape $eq(E, r)$, for a path expression E and a property r , tests equality of the sets of nodes reachable from the focus node by an r -edge on the one hand, and by an E -path on the other hand. A similar shape $disj(E, r)$ tests disjointness of the two sets. To illustrate, example shape 4 is expressed as $\neg disj(\text{colleague}, \text{friend})$.

Closure constraints: RDF graphs to be checked for conformance against some shape schema need not obey some fixed vocabulary. Thus SHACL provides shapes of the form $closed(R)$, with R a finite set of properties, expressing that the focus node has no properties other than those in R . This was already illustrated as example shape 5, with $R = \{\text{name}, \text{address}, \text{birthdate}\}$.

Zero-or-one paths: If E is a path expression, then the path expression $E?$, evaluated in a focus node v , yields the set of nodes reachable from v by an E -path, plus v itself. This constructor, which is also present in the RDF query language SPARQL, is a very special case of *tests* in the logic PDL [7], where $E?$ would be written as $E \cup \top?$. For example, in a function call graph as used in software engineering, the shape $\geq_5 \text{calls}?.\top$ expresses that the focus node calls at least *four* functions other than itself. Interestingly, zero-or-one paths allow the expression of self-restriction, a feature introduced in the logic SROIQ [11] and adopted in the Web ontology language [14]. For example, staying with the function calls, to express in SROIQ that the focus node calls itself, one can write $\exists \text{calls}.\text{Self}$. As a SHACL shape, we can write $eq(\text{calls}?, \text{calls})$.

Our goal in this paper is to clarify the impact of the above four features on the expressiveness of SHACL as a language for boolean queries on graph databases. Thereto, we offer the following contributions.

- We show that each of the four features is primitive in a strong sense. Specifically, for each feature, we exhibit a boolean query Q such that Q is expressible by a single target–shape pair, using only the feature and the basic constructs; however, Q is not expressible by any generalized shape schema when the feature is disallowed.
- We also clarify the significance of the restriction that SHACL puts on allowed targets. We observe that as long as closure constraints are not used, the restriction is actually insignificant. Any generalized shape schema, allowing arbitrary but closure-free shapes on the left-hand sides of the inclusion statements, can be equivalently written as a shape schema with only targets on the left-hand sides. However, allowing closure constraints on the left-hand side strictly adds expressive power.
- Our results continue to hold when the definition of *recursive* shapes is allowed, provided that recursion through negation is stratified.

This paper is organized as follows. Section 2 presents clean formal definitions of non-recursive shape schemas, building on and inspired by the work of Andreşel, Cormann, et al. cited above. Section 3 presents our results. Section 4 presents the extension to stratified recursion. Section 5 offers concluding remarks.

2 Shape schemas

In this section we define shapes, RDF graphs, shape schemas, and the conformance of RDF graphs to shape schemas. Perhaps curiously to those familiar with SHACL, our treatment for now omits *shape names*. Shape names are redundant as far as expressive power is concerned, as long as we are in a non-recursive setting, because shape name definitions can then always be unfolded. Indeed, for clarity of exposition, we have chosen to work first with non-recursive shape schemas. Section 4 then presents the extension to recursion (and introduces shape names in the process). We point out that the W3C SHACL recommendation only considers non-recursive shape schemas.

Node and property names

From the outset we assume two disjoint, infinite universes N and P of *node names* and *property names*, respectively.³

2.1 Shapes

We define *path expressions* E and *shapes* ϕ by the following grammar:

$$\begin{aligned} E &::= p \mid p^- \mid E \cup E \mid E/E \mid E? \mid E^* \\ \phi &::= \top \mid \{c\} \mid \phi \wedge \phi \mid \phi \vee \phi \mid \neg\phi \mid \geq_n E.\phi \mid eq(E, p) \mid disj(E, p) \mid closed(R) \end{aligned}$$

Here, p and c stand for property names and node names, respectively; n stands for nonzero natural numbers; and R stands for finite sets of property names. A node name c is also referred to as a *constant*.

Abbreviation: We will use $\exists E.\phi$ as an abbreviation for $\geq_1 E.\phi$.

► **Remark 1.** Real SHACL supports some further shapes which have to do with tests on IRI constants and literals, as well as comparisons on numerical values and language tags. Like other work on the formal aspects of SHACL, we abstract these away, as many questions are already interesting without these features.

► **Remark 2.** Universal quantification $\forall E.\phi$ could be introduced as an abbreviation for $\neg\exists E.\neg\phi$. Likewise, $\leq_n E.\phi$ may be used as an abbreviation for $\neg\geq_{n+1} E.\phi$.

A *vocabulary* Σ is a subset of $N \cup P$. A path expression is said to be *over* Σ if it only uses property names from Σ . Likewise, a shape is *over* Σ if it only uses constants from Σ and path expressions over Σ .

Following common practice in logic, shapes are evaluated in *interpretations*. We recall the familiar definition of an interpretation. Let Σ be a vocabulary. An interpretation I over Σ consists of

- a set Δ^I , called the *domain* of I ;
- for each constant $c \in \Sigma$, an element $\llbracket c \rrbracket^I \in \Delta^I$; and
- for each property name $p \in \Sigma$, a binary relation $\llbracket p \rrbracket^I$ on Δ^I .

The semantics of shapes is now defined as follows.

³ In practice, node names and property names are IRIs [16], so the disjointness assumption would not hold. However, this assumption is only made for simplicity of notation; it can be avoided if we make our notation for vocabularies and interpretations (see below) more complicated.

■ **Table 1** Semantics of path expressions.

E	$\llbracket E \rrbracket^I$
p^-	$\{(a, b) \mid (b, a) \in \llbracket p \rrbracket^I\}$
$E_1 \cup E_2$	$\llbracket E_1 \rrbracket^I \cup \llbracket E_2 \rrbracket^I$
E_1/E_2	$\{(a, b) \mid \exists x : (a, x) \in \llbracket E_1 \rrbracket^I \wedge (x, b) \in \llbracket E_2 \rrbracket^I\}$
$E?$	$\llbracket E \rrbracket^I \cup \{(x, x) \mid x \in \Delta^I\}$
E^*	the reflexive-transitive closure of $\llbracket E \rrbracket^I$

■ **Table 2** Conditions for conformance of a node to a shape.

ϕ	$I, a \models \phi$ if:
$\{c\}$	$a = \llbracket c \rrbracket^I$
$\geq_n E.\psi$	$\#\{b \in \llbracket E \rrbracket^I(a) \mid I, b \models \psi\} \geq n$
$eq(E, p)$	the sets $\llbracket E \rrbracket^I(a)$ and $\llbracket p \rrbracket^I(a)$ are equal
$disj(E, p)$	the sets $\llbracket E \rrbracket^I(a)$ and $\llbracket p \rrbracket^I(a)$ are disjoint
$closed(R)$	$\llbracket p \rrbracket^I(a)$ is empty for each $p \in \Sigma - R$

- On any interpretation I as above, every path expression E over Σ evaluates to a binary relation $\llbracket E \rrbracket^I$ on Δ^I , defined in Table 1.
- Now for any shape ϕ over Σ and any element $a \in \Delta^I$, we define when a *conforms to ϕ in I* , denoted by $I, a \models \phi$. For the boolean operators \top (true), \wedge (conjunction), \vee (disjunction), \neg (negation), the definition is obvious. For the other constructs, the definition is given in Table 2, taking note of the following:
 - We use the notation $R(x)$, for a binary relation R , to denote the set $\{y \mid (x, y) \in R\}$. We apply this notation to the case where R is of the form $\llbracket E \rrbracket^I$.
 - We also use the notation $\#X$ for the cardinality of a set X .
- For a shape ϕ and interpretation I , the notation

$$\llbracket \phi \rrbracket^I := \{a \in \Delta^I \mid I, a \models \phi\}$$

will be convenient.

2.2 Graphs and their interpretation

It may appear that a shape $closed(R)$ is simply expressible as the conjunction of $\neg \exists p. \top$ for $p \in \Sigma - R$. However, since shapes must be finite formulas, this only works if Σ is finite. In practice, shapes can be evaluated over arbitrary RDF graphs, which can involve arbitrary property names (and node names), not limited to a finite vocabulary that is fixed in advance.

Formally, we define a *graph* as a finite set of triples of the form (a, p, b) , where p is a property name and a and b are (not necessarily distinct) node names. We refer to the node names appearing in a graph G simply as the *nodes* of G ; the set of nodes of G is denoted by N_G . A pair (a, b) with $(a, p, b) \in G$ is referred to as an *edge*, or a *p-edge*, in G .

We now canonically view any graph G as an interpretation over the *full* vocabulary $N \cup P$ as follows:

- Δ^G equals N (the universe of all node names).
- $\llbracket c \rrbracket^G$ equals c itself, for every node name c .
- $\llbracket p \rrbracket^G$ equals the set of p -edges in G , for every property name p .

Note since graphs are finite, $\llbracket p \rrbracket^G$ will be empty for all but a finite number of p 's.

Given this canonical interpretation, path expressions and shapes obtain a semantics on all graphs G . Thus for any path expression E , the binary relation $\llbracket E \rrbracket^G$ on N is well-defined; for any shape ϕ and $a \in N$, it is well-defined whether or not $G, a \models \phi$; and we can also use the notation $\llbracket \phi \rrbracket^G$.

► **Remark 3.** Since a graph is considered to be an interpretation with the infinite domain N , it may not be immediately clear that shapes can be effectively evaluated over graphs. Adapting well-known methods, however, we can reduce to a finite domain over a finite vocabulary [1, Theorem 5.6.1], [3, 12]. Formally, let ϕ be a shape and let G be a graph. Recall that N_G denotes the set of nodes of G ; similarly, let P_G be the set of property names appearing in G . Let C be the set of constants mentioned in ϕ . We can then form the finite vocabulary $\Sigma = N_G \cup C \cup P_G$. Now define the interpretation I over Σ as follows:

- $\Delta^I = N_G \cup C \cup \{\star\}$, where \star is an element not in N ;
- $\llbracket c \rrbracket^I = c$ for each node name $c \in \Sigma$;
- $\llbracket p \rrbracket^I = \llbracket p \rrbracket^G$ for each property name $p \in \Sigma$.

Note that no constant symbol names \star in I . Then for every $x \in N_G \cup C$, one can show that $x \in \llbracket \phi \rrbracket^G$ if and only if $x \in \llbracket \phi \rrbracket^I$. For all other node names x , one can show that $x \in \llbracket \phi \rrbracket^G$ if and only if $\star \in \llbracket \phi \rrbracket^I$.

In our companion paper [5] we argue why using the infinite domain N is the most natural approach.

2.3 Targets and shape schemas

SHACL identifies four special forms of shapes and calls them *targets*:

Node targets: $\{c\}$ for any constant c .

Class-based targets: $\exists \text{type/subclass}^*.\{c\}$ for any constant c . Here, *type* and *subclass* represent distinguished IRIs from the RDF Schema vocabulary [16].

Subjects-of targets: $\exists p.\top$ for any property name p .

Objects-of targets: $\exists p^*.\top$ for any property name p .

We now define a *generalized shape schema* (or shape schema for short) as a finite set of inclusion statements, where an inclusion statement is of the form $\phi_1 \subseteq \phi_2$, with ϕ_1 and ϕ_2 shapes. A *target-based shape schema* is a shape schema that only uses targets, as defined above, on the left-hand sides of its inclusion statements. This restriction corresponds to the shape schemas considered in real SHACL.

As already explained in the Introduction, a graph G *conforms* to a shape schema \mathcal{S} , denoted by $G \models \mathcal{S}$, if $\llbracket \phi_1 \rrbracket^G$ is a subset of $\llbracket \phi_2 \rrbracket^G$, for every statement $\phi_1 \subseteq \phi_2$ in \mathcal{S} .

Thus, any shape schema \mathcal{S} defines the class of graphs that conform to it. We denote this class of graphs by

$$\llbracket \mathcal{S} \rrbracket := \{\text{graph } G \mid G \models \mathcal{S}\}.$$

Accordingly, two shape schemas \mathcal{S}_1 and \mathcal{S}_2 are said to be *equivalent* if $\llbracket \mathcal{S}_1 \rrbracket = \llbracket \mathcal{S}_2 \rrbracket$.

3 Expressiveness of SHACL features

When a complicated but influential new tool is proposed in the community, in our case SHACL, we feel it is important to have a solid understanding of its design. Concretely, as motivated in the Introduction, our goal is to obtain a clear picture of the relative expressiveness of the features *eq*, *disj*, *closed*, and *?*. Our methodology is as follows.

A *feature set* F is a subset of $\{eq, disj, closed, ?\}$. The set of all shape schemas using only features from F , besides the standard constructs, is denoted by $\mathcal{L}(F)$. In particular, shape schemas in $\mathcal{L}(\emptyset)$ use only the standard constructs and none of the four features. Specifically, they only involve shapes built from boolean connectives, constants, and qualified number restrictions, with path expressions built from the standard operators union, composition, and Kleene star.

We say that feature set F_1 is *subsumed* by feature set F_2 , denoted by $F_1 \preceq F_2$, if every shape schema in $\mathcal{L}(F_1)$ is equivalent to some shape schema in $\mathcal{L}(F_2)$. As it will turn out,

$$F_1 \preceq F_2 \quad \Leftrightarrow \quad F_1 \subseteq F_2, \quad (*)$$

or intuitively, “every feature counts.” Note that the implication from right to left is trivial, but the other direction is by no means clear from the outset.

More specifically, for every feature, we introduce a class of graphs, as follows. In what follows we fix some property name r .

Equality: Q_{eq} is the class of graphs where all r -edges are symmetric. Note that Q_{eq} is definable in $\mathcal{L}(eq)$ by the single, target-based, inclusion statement $\exists r. \top \subseteq eq(r^-, r)$.

Disjointness: Q_{disj} is the class of graphs where all nodes with an outgoing r -edge have at least one symmetric r -edge. This time, Q_{disj} is definable in $\mathcal{L}(disj)$, by the single, target-based, inclusion statement $\exists r. \top \subseteq \neg disj(r^-, r)$.

Closure: Q_{closed} is the class of graphs where for all nodes with an outgoing r -edge, all outgoing edges have label r . Again Q_{closed} is definable in $\mathcal{L}(closed)$ by the single, target-based, inclusion statement $\exists r. \top \subseteq closed(r)$.

Zero-or-one path: Finally, $Q_?$ is the class of graphs where all nodes with an outgoing r -edge have at least three outgoing r -edges *not* to themselves (i.e., not counting possible self-loops). As expected $Q_?$ is definable in $\mathcal{L}(?)$ by the single, target-based, inclusion statement $\exists r. \top \subseteq \geq_4 r?.\top$.

We establish the following theorem, from which the above equivalence (*) immediately follows:

► **Theorem 4.** *Let $X \in \{eq, disj, closed, ?\}$ and let F be a feature set with $X \notin F$. Then Q_X is not definable in $\mathcal{L}(F)$.*

3.1 Equality, disjointness, and zero-or-one path

For $X = closed$, Theorem 4 is proven differently than for the other three features. We deal with *closed* in the next subsection. Here, we deal with the remaining features through the following concrete result, illustrated in Figure 1:

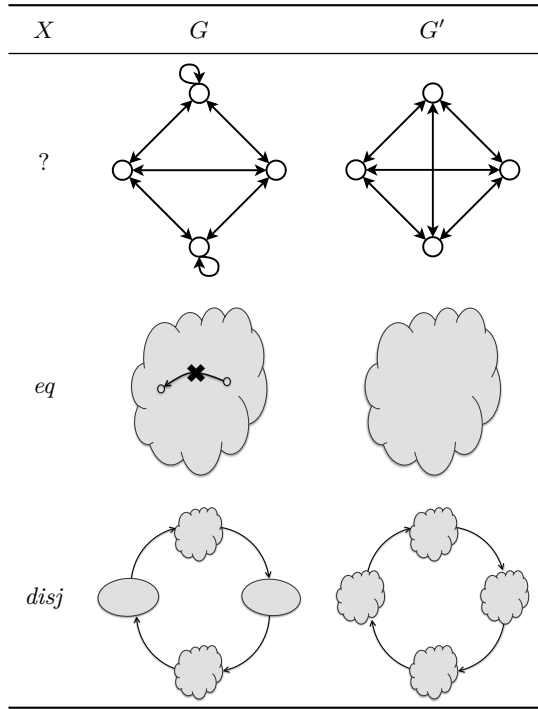
► **Proposition 5.** *Let $X = eq, disj, \text{ or } ?$. Let Σ be a finite vocabulary including r , and let m be a nonzero natural number. There exist two graphs G and G' with the following properties:*

1. G' belongs to Q_X , but G does not.
2. For every shape ϕ over Σ such that ϕ does not use X , and ϕ counts to at most m , we have

$$\llbracket \phi \rrbracket^G = \llbracket \phi \rrbracket^{G'}.$$

Here, “counting to at most m ” means that all quantifiers \geq_n used in ϕ satisfy $n \leq m$.

To see that Proposition 5 indeed establishes Theorem 4 for the three features under consideration, we use the notion of *validation shape* of a shape schema. This shape evaluates to the set of all nodes that violate the schema. Thus, the validation shape is an abstraction



■ **Figure 1** Graphs used to prove Proposition 5. The nodes are taken outside Σ . For $X = eq$, the cloud shown for G' represents a complete directed graph on $m + 1$ nodes, with self-loops, and G is the same graph with one directed edge removed. For $X = disj$, in the picture for G , each cloud again stands for a complete graph, but this time on $M = \max(m, 3)$ nodes, and without the self-loops. Each oval stands for a set of M separate nodes. An arrow from one blob to the next means that every node of the first blob has a directed edge to every node of the next blob. So, G is a directed 4-cycle of alternating clouds and ovals, and G' is a directed 4-cycle of clouds.

of the “validation report” in SHACL [18]: a graph conforms to a schema if and only if the validation shape evaluates to the empty set. The validation shape can be formally constructed as the disjunction of $\phi_1 \wedge \neg\phi_2$ for all statements $\phi_1 \subseteq \phi_2$ in the schema.

Now consider a shape schema \mathcal{S} not using feature X . Let m be the maximum count used in shapes in \mathcal{S} , and let Σ' be the set of constants and property names mentioned in \mathcal{S} . Now given $\Sigma = \Sigma' \cup \{r\}$ and m , let G and G' be the two graphs exhibited by the Proposition, and let ϕ be the validation shape for \mathcal{S} . Then ϕ will evaluate to the same result on G and G' . However, for \mathcal{S} to define Q_X , validation would have to return the empty set on G' but a nonempty set on G . We conclude that \mathcal{S} does not define Q_X .

Proof of Proposition 5

We present here the proof for $X = disj$. The general strategy is to first characterize the behavior of path expressions on G and G' . Then the Proposition is proven with a stronger induction hypothesis, to allow the induction to carry through. A similar strategy is followed in the proofs for $X = eq$ and $X = ?$.

We begin by defining the graphs G and G' more formally.

► **Definition 6** ($G_{disj}(\Sigma, m)$). Let Σ be a finite vocabulary including r , and let m be a natural number. We define the graph $G_{disj}(\Sigma, m)$ over the set of property names in Σ as follows. Let $M = \max(m, 3)$. There are $4M$ nodes in the graph, which are chosen outside of Σ . We

denote these nodes by x_i^j for $i = 1, 2, 3, 4$ and $j = 1, \dots, M$. (In the description that follows, subscripts range from 1 to 4 and superscripts range from 1 to M .) For each property name p in Σ , the graph has the same set of p -edges. There is an edge from x_i^j to $x_{i \bmod 4 + 1}^{j'}$ for every i, j and j' . Moreover, if i is 2 or 4, there is an edge from x_i^j to $x_i^{j'}$ for all $j \neq j'$.

Thus, in Figure 1, bottom left, one can think of the left oval as the set of nodes x_1^j ; the top cloud as the set of nodes x_2^j ; and so on. We call the nodes x_i^j with $i = 2, 4$ the *even nodes*, and the nodes x_i^j with $i = 1, 3$ the *odd nodes*.

► **Definition 7** ($G'_{disj}(\Sigma, m)$). We define the graph $G'_{disj}(\Sigma, m)$ in the same way as $G_{disj}(\Sigma, m)$ except that there is an edge from x_i^j to $x_i^{j'}$ for all i and $j \neq j'$ (not only for i even).

We characterize the behavior of path expressions on the graph $G_{disj}(\Sigma, m)$ as follows.

► **Lemma 8.** Let G be $G_{disj}(\Sigma, m)$. Call a path expression simple if it is a union of property names. Let E be a non-simple path expression over Σ . The following three statements hold:

1. **A.** for all even nodes v of G , we have $\llbracket E \rrbracket^G(v) \supseteq \llbracket r \rrbracket^G(v)$; or
- B.** for all even nodes v of G , we have $\llbracket E \rrbracket^G(v) \supseteq \llbracket r^- \rrbracket^G(v)$.
2. **C.** for all odd nodes v of G , we have $\llbracket E \rrbracket^G(v) \supseteq \llbracket r \rrbracket^G(v)$; or
- D.** for all odd nodes v of G , we have $\llbracket E \rrbracket^G(v) \supseteq \llbracket r^- \rrbracket^G(v)$.
3. For all nodes v of G , we have $\llbracket E \rrbracket^G(v) - \llbracket r \rrbracket^G(v) \neq \emptyset$.

Proof. We use the notation $X_i = \{x_i^1, \dots, x_i^M\}$ for the i th blob of nodes. We also use the notations $next(1) = 2$; $next(2) = 3$; $next(3) = 4$; $next(4) = 1$; $prev(4) = 3$; $prev(3) = 2$; $prev(2) = 1$; $prev(1) = 4$. Thus $next(i)$ indicates the next blob in the cycle, and $prev(i)$ the previous.

The proof is by induction on the structure of E . If E is a property name, E is simple so the claim is trivial. If E is of the form p^- , cases B and D are clear and we only need to verify the third statement. That holds because for any i , if $v \in X_i$, then $\llbracket p^- \rrbracket^G(v) \supseteq X_{prev(i)}$ and clearly $X_{prev(i)} - \llbracket r \rrbracket^G(v) \neq \emptyset$. We next consider the inductive cases.

First, assume E is of the form $E_1 \cup E_2$. When at least one of E_1 and E_2 is not simple, the three statements immediately follow by induction, since $\llbracket E \rrbracket^G \supseteq \llbracket E_1 \rrbracket^G$ and $\llbracket E \rrbracket^G \supseteq \llbracket E_2 \rrbracket^G$. If E_1 and E_2 are simple, then E is simple and the claim is trivial.

Next, assume E is of the form $E_1?$ or E_1^* . If E_1 is not simple, the three statements follow immediately by induction, since $\llbracket E \rrbracket^G \supseteq \llbracket E_1 \rrbracket^G$. If E_1 is simple, cases A and C clearly hold for E , so we only need to verify the third statement. That holds because, by the form of E , every node v is in $\llbracket E \rrbracket^G(v)$, but not in $\llbracket r \rrbracket^G(v)$, as G does not have any self-loops.

Finally, assume E is of the form E_1/E_2 . Note that if E_1 or E_2 is simple, clearly cases A and C apply to them. The argument that follows will therefore also apply when E_1 or E_2 is simple. We will be careful not to apply the induction hypothesis for the third statement to E_1 and E_2 .

We first focus on the even nodes, and show the first and the third statement. We distinguish two cases.

- If case A applies to E_2 , then we show that case A also applies to E . Let $v \in X_i$ be an even node. We verify the following two inclusions:
 - $\llbracket E \rrbracket^G(v) \supseteq X_i$. Let $u \in X_i$. If $u \neq v$, choose a third node $w \in X_i$. Since X_i is a clique, $(v, w) \in \llbracket E_1 \rrbracket^G$ regardless of whether case A or B applies to E_1 . By case A for E_2 , we also have $(w, u) \in \llbracket E_2 \rrbracket^G$, whence $u \in \llbracket E \rrbracket^G(v)$ as desired. If $u = v$, we similarly have $(v, w) \in \llbracket E_1 \rrbracket^G$ and $(w, u) \in \llbracket E_2 \rrbracket^G$ as desired.

15:10 Expressiveness of SHACL Features

- $\llbracket E \rrbracket^G(v) \supseteq X_{next(i)}$. Let $u \in X_{next(i)}$ and choose $w \neq v \in X_i$. Regardless of whether case A or B applies to E_1 , we have $(v, w) \in \llbracket E_1 \rrbracket^G$. By case A for E_2 , we also have $(w, u) \in \llbracket E_2 \rrbracket^G$, whence $u \in \llbracket E \rrbracket^G(v)$ as desired.

We conclude that $\llbracket E \rrbracket^G(v) \supseteq X_i \cup X_{next(i)} \supseteq \llbracket r \rrbracket^G$ as desired.

- If case B applies to E_2 , then we show that case B also applies to E . This is analogous to the previous case, now verifying that $\llbracket E \rrbracket^G(v) \supseteq X_i \cup X_{prev(i)}$.

In both cases, the third statement now follows for even nodes v . Indeed, $v \in X_i \subseteq \llbracket E \rrbracket^G(v)$ but $v \notin \llbracket r \rrbracket^G(v)$.

We next focus on the odd nodes, and show the second and the third statement. We again consider two cases.

- If case C applies to E_1 , then we show that case C also applies to E . Let $v \in X_i$ be an odd node. Note that $\llbracket r \rrbracket^G(v) = X_{next(i)}$. To verify that $\llbracket E \rrbracket^G(v) \supseteq X_{next(i)}$, let $u \in X_{next(i)}$. Then u is even. Choose $w \neq u \in X_{next(i)}$. Since case C applies to E_1 , we have $(v, w) \in \llbracket E_1 \rrbracket^G$. Moreover, since $X_{next(i)}$ is a clique, $(w, u) \in \llbracket E_2 \rrbracket^G$ regardless of whether case A or B applies to E_2 . We obtain $(v, u) \in \llbracket E \rrbracket^G$ as desired.

We also verify the third statement for odd nodes in this case. We distinguish two further cases.

- If case A applies to E_2 , any node $u \in X_{next(next(i))}$ belongs to $\llbracket E \rrbracket^G(v)$, and clearly these u are not in $X_{next(i)} = \llbracket r \rrbracket^G(v)$.
- If case B applies to E_2 , then, since X_i is a clique, any node $u \in X_i$ belongs to $\llbracket E \rrbracket^G(v)$, and again these u are not in $X_{next(i)}$.
- If case D applies to E_1 , then we show that case D also applies to E . This is analogous to the previous case, now verifying that $\llbracket E \rrbracket^G(v) \supseteq X_{prev(i)}$. In this case the third statement for odd nodes is clear, as clearly $X_{prev(i)} - X_{next(i)} \neq \emptyset$. ◀

We similarly characterize the behavior of path expressions on the other graph. The characterization is simpler to state and simpler to verify, due to the homogeneous nature of the graph $G'_{disj}(\Sigma, m)$. We omit the proof.

► **Lemma 9.** *Let G' be $G'_{disj}(\Sigma, m)$ and let E be a non-simple path expression over Σ . The following statements hold:*

1. $\llbracket E \rrbracket^{G'} \supseteq \llbracket r \rrbracket^{G'}$ or $\llbracket E \rrbracket^{G'} \supseteq \llbracket r^- \rrbracket^{G'}$.
2. For all nodes v of G' , we have $\llbracket E \rrbracket^{G'}(v) - \llbracket r \rrbracket^{G'}(v) \neq \emptyset$.

We also need the following Definition and Lemma, detailing how path expressions can behave on the nodes outside a graph.

► **Definition 10 (Safety).** *We define the safety of a path expression E inductively as follows:*

- If E is p or p^- , then E is safe.
- If E is $E_1 \cup E_2$, then E is safe only if E_1 and E_2 are safe.
- If E is E_1/E_2 , then E is safe if E_1 or E_2 is safe.
- If E is E_1^* or $E^?$, then E is not safe.

► **Lemma 11.** *Let E be a path expression and let G be a graph.*

- If E is safe, then $\llbracket E \rrbracket^G \subseteq N_G \times N_G$.
- If E is not safe, then $\llbracket E \rrbracket^G = (\llbracket E \rrbracket^G \cap N_G \times N_G) \cup \{(a, a) \mid a \in N - N_G\}$.

We are now ready to prove the non-obvious part of Proposition 5, in the following version.

► **Proposition 5 for $X = \text{disj}$, stronger version.** Let V be the common set of nodes of the graphs $G = G_{\text{disj}}(\Sigma, m)$ and $G' = G'_{\text{disj}}(\Sigma, m)$. Let ϕ be a shape over Σ that does not use disj , and that maximally counts to m . Then either $\llbracket \phi \rrbracket^G \cap V = \emptyset$ or $\llbracket \phi \rrbracket^G \supseteq V$. Moreover, $\llbracket \phi \rrbracket^G = \llbracket \phi \rrbracket^{G'}$.

This is proven by induction on the structure of ϕ . Let H be G or G' . If ϕ is \top , then $\llbracket \top \rrbracket^H = N \supseteq V$. If ϕ is $\{c\}$, then $\llbracket \{c\} \rrbracket^H = \{c\} \subseteq \Sigma$ and we know that $\Sigma \cap V = \emptyset$. We next consider the inductive cases. The cases for the boolean connectives follow readily by induction.

Now assume ϕ is of the form $\geq_n E.\phi_1$. By induction, there are two possibilities for ϕ_1 :

- If $\llbracket \phi_1 \rrbracket^H \cap V = \emptyset$, then also $\llbracket \phi \rrbracket^H \cap V = \emptyset$ since path expressions can only reach nodes in some graph from nodes in that graph.
- If $\llbracket \phi_1 \rrbracket^H \supseteq V$, to show that $\llbracket \phi \rrbracket^H \supseteq V$, it suffices to show that $\sharp \llbracket E \rrbracket^H(v) \geq n$ for all $v \in V$. By Lemmas 8 and 9 we know that $\llbracket E \rrbracket^H(v)$ contains $\llbracket r \rrbracket^H(v)$ or $\llbracket r^- \rrbracket^H(v)$. Inspecting H , we see that each of these sets has at least $\max(3, m) \geq n$ elements, as desired.

In both cases we still need to show that $\llbracket \phi \rrbracket^G = \llbracket \phi \rrbracket^{G'}$. We already showed that $\llbracket \phi \rrbracket^G \supseteq V$ and $\llbracket \phi \rrbracket^{G'} \supseteq V$, or $\llbracket \phi \rrbracket^G \cap V = \emptyset$ and $\llbracket \phi \rrbracket^{G'} \cap V = \emptyset$. Therefore, towards a proof of the equality, we only need to consider the node names not in V .

For the inclusion from left to right, take $x \in \llbracket \phi \rrbracket^G - V$. Since $G, x \models \phi$, there exists y_1, \dots, y_n such that $(x, y_i) \in \llbracket E \rrbracket^G$ and $G, y_i \models \phi_1$ for $i = 1, \dots, n$. However, since $x \notin V$, by Lemma 11, all y_i must equal x . Hence, $n = 1$ and $(x, x) \in \llbracket E \rrbracket^G$ and $G, x \models \phi_1$. Then again by the same Lemma, $(x, x) \in \llbracket E \rrbracket^{G'}$, since G and G' have the same set of nodes V . Moreover, by induction, $G', x \models \phi_1$. We conclude that $G', x \models \phi$ as desired. The inclusion from right to left is argued symmetrically.

Next assume ϕ is of the form $\text{eq}(E, p)$. If E is simple then $\llbracket E \rrbracket^H = \llbracket p \rrbracket^H$, so clearly $\llbracket \phi \rrbracket^H = N \supseteq V$.

If E is not simple, then Lemmas 8 and 9 tell us that $\llbracket E \rrbracket^H(v) - \llbracket r \rrbracket^H(v) \neq \emptyset$ for every $v \in V$. Since $\llbracket r \rrbracket^H = \llbracket p \rrbracket^H$, this means $H, v \not\models \phi$ for $v \in V$, or, equivalently, $\llbracket \phi \rrbracket^G \cap V = \emptyset$. To see that, moreover, $\llbracket \phi \rrbracket^G = \llbracket \phi \rrbracket^{G'}$, it remains to show that $G, v \models \phi$ iff $G', v \models \phi$ for all node names $v \notin V$. Clearly, $\llbracket p \rrbracket^G(v) = \llbracket p \rrbracket^{G'}(v) = \emptyset$. Now by Lemma 11, if E is safe, then also $\llbracket E \rrbracket^G(v) = \llbracket E \rrbracket^{G'}(v) = \emptyset$, so $G, v \models \phi$ and $G', v \models \phi$. On the other hand, if E is unsafe, then by the same Lemma $\llbracket E \rrbracket^G(v) = \llbracket E \rrbracket^{G'}(v) = \{v\} \neq \emptyset$, so $G, v \not\models \phi$ and $G', v \not\models \phi$, as desired.

Finally, assume ϕ is of the form $\text{closed}(R)$. If Σ contains a property name p not in R , then $\llbracket \phi \rrbracket^H \cap V = \emptyset$, since every node in H has an outgoing p -edge. Otherwise, i.e., if $\Sigma \subseteq R$, we have $\llbracket \phi \rrbracket^H \supseteq V$, since every node in H has only outgoing edges labeled by property names in Σ . To see that, moreover, $\llbracket \phi \rrbracket^G = \llbracket \phi \rrbracket^{G'}$, it suffices to observe that trivially $H, v \models \phi$ for all node names $v \notin V$.

3.2 Closure

Without using closed , shapes cannot say anything about properties that they do not explicitly mention. We formalize this intuitive observation as follows. The proof is straightforward.

► **Lemma 12.** Let Σ be a vocabulary, let E be a path expression over Σ , and let ϕ be a shape over Σ that does not use closed . Let G_1 and G_2 be graphs such that $\llbracket p \rrbracket^{G_1} = \llbracket p \rrbracket^{G_2}$ for every property name p in Σ . Then $\llbracket E \rrbracket^{G_1} = \llbracket E \rrbracket^{G_2}$ and $\llbracket \phi \rrbracket^{G_1} = \llbracket \phi \rrbracket^{G_2}$.

Theorem 4 now follows readily for $X = \text{closed}$. Let F be a feature set without closed , let \mathcal{S} be a shape schema in $\mathcal{L}(F)$, and let ϕ be the validation shape of \mathcal{S} . Let p be a property name not mentioned in \mathcal{S} , and different from r . Consider the graphs $G = \{(a, r, a), (a, p, a)\}$ and $G' = \{(a, r, a)\}$, so that G' belongs to Q_{closed} but G does not. By Lemma 12 we have $\llbracket \phi \rrbracket^G = \llbracket \phi \rrbracket^{G'}$, showing that \mathcal{S} does not define Q_{closed} .

► Remark 13. Lemma 12 fails completely in the presence of closure constraints. The simplest counterexample is to consider $\Sigma = \emptyset$ and the shape $closed(\emptyset)$. Trivially, any two graphs agree on the property names from Σ . However, $\llbracket closed(\emptyset) \rrbracket^G$, which equals the set of node names that do not have an outgoing edge in G (they may still have an incoming edge), obviously depends on the graph G .

3.3 Are target-based shape schemas enough?

Lemma 12 also allows us to clarify that, as far as expressive power is concerned, and in the absence of closure constraints, the restriction to target-based shape schemas is inconsequential.

► **Theorem 14.** *Every generalized shape schema that does not use closure constraints is equivalent to a target-based shape schema (that still does not use closure constraints).*

Proof. Let ϕ be the validation shape for shape schema \mathcal{S} , so that $G \models \mathcal{S}$ if and only if $\llbracket \phi \rrbracket^G$ is empty. Let C be the set of constants mentioned in ϕ .

Let us say that ϕ is *internal* if for every graph G and every node name v such that $G, v \models \phi$, we have $v \in N_G \cup C$. If ϕ is not internal, it can be shown that for every graph G and every node $v \notin N_G \cup C$, we have $G, v \models \phi$. Thus, if ϕ is not internal, \mathcal{S} is unsatisfiable and is equivalent to the single target-based inclusion $\{c\} \subseteq \neg \top$, for an arbitrary constant c .

So now assume ϕ is internal. Define the target-based shape schema \mathcal{T} consisting of the following inclusions:

- For each constant $c \in C$, the inclusion $\{c\} \subseteq \neg \phi$.
 - For each property name mentioned in ϕ , the two inclusions $\exists p. \top \subseteq \neg \phi$ and $\exists p^-. \top \subseteq \neg \phi$.
- We will show that \mathcal{S} and \mathcal{T} are equivalent. Let ψ be the validation shape for \mathcal{T} .

Let G be any graph, and let G' be the graph obtained from G by removing all triples involving property names not mentioned in ϕ . We reason as follows:

$$\begin{aligned}
 G \models \mathcal{S} &\Leftrightarrow \llbracket \phi \rrbracket^G = \emptyset \\
 &\Leftrightarrow \llbracket \phi \rrbracket^{G'} = \emptyset && \text{by Lemma 12} \\
 &\Leftrightarrow G' \models \mathcal{T} && \text{since } \phi \text{ is internal} \\
 &\Leftrightarrow \llbracket \psi \rrbracket^{G'} = \emptyset \\
 &\Leftrightarrow \llbracket \psi \rrbracket^G = \emptyset && \text{by Lemma 12} \\
 &\Leftrightarrow G \models \mathcal{T} \quad \blacktriangleleft
 \end{aligned}$$

► Remark 15. Note that we do not need class-based targets in the proof, so such targets are redundant on the left-hand sides of inclusions. This can also be seen directly: any inclusion

$$\exists \text{type}/\text{subclass}^*. \{c\} \subseteq \phi$$

with a class-based target is equivalent to the following inclusion with a subjects-of target:

$$\exists \text{type}. \top \subseteq \neg \exists \text{type}/\text{subclass}^*. \{c\} \vee \phi$$

► Remark 16. Theorem 14 fails in the presence of closure constraints. For example, the inclusion $\neg closed(\emptyset) \subseteq \exists r. \top$ defines the class of graphs where every node with an outgoing edge has an outgoing r -edge. Suppose this inclusion would be equivalent to a target-based shape schema \mathcal{S} , and let R be the set of all property names mentioned in the targets of \mathcal{S} . Let p be a property name not in R and distinct from r ; let a be a node name not used as a constant in \mathcal{S} ; and consider the graph $G = \{(a, p, a)\}$. This graph trivially satisfies \mathcal{S} , but violates the inclusion.

4 Extension to stratified recursion

Until now, we could do without shape names. We do need them, however, for recursive shape schemas. Such schemas allow shapes to be defined using recursive rules, much as in Datalog and logic programming. The rules have a shape name in the head; in the body they have a shape that can refer to the same or other shape names.

► **Example 17.** The following rule defines a shape, named s , recursively:

$$s \leftarrow \{c\} \vee (eq(p, q) \wedge \exists r.s).$$

A node x will satisfy s if there is a path of r -edges from x to the constant c , so that all nodes along the path satisfy $eq(p, q)$ (for two property names p and q).

Rules and programs

We need to make a few extensions to our formalism and the semantics.

- We assume an infinite supply S of *shape names*. Again for simplicity of notation only, we assume that S is disjoint from N and P .
- The syntax of shapes is extended so that *every shape name is a shape*.
- A vocabulary Σ is now a subset of $N \cup P \cup S$; an interpretation I now additionally assigns a subset $\llbracket s \rrbracket^I$ of Δ^I to every shape name s in Σ .

Noting the obvious parallels with the field of logic programming, we propose to use the following terminology from that field. A *rule* is of the form $s \leftarrow \phi$, where s is a shape name and ϕ is a shape. A *program* is a finite set of rules. The shape names appearing as heads of rules in a program are called the *intensional* shape names of that program.

The following definitions of the semantics of programs are similar to definitions well-known for Datalog. A program is *semipositive* if for every intensional shape name s , and every shape ϕ in the body of some rule, s occurs only positively in ϕ . Let \mathcal{P} be a semipositive program over vocabulary Σ , with set of intensional shape names D . An interpretation J over $\Sigma \cup D$ is called a *model* of \mathcal{P} if for every rule $s \leftarrow \phi$ of \mathcal{P} , the set $\llbracket \phi \rrbracket^J$ is a subset of $\llbracket s \rrbracket^J$. Given any interpretation I over $\Sigma - D$, there exists a unique minimal interpretation J that expands I to $\Sigma \cup D$ such that J is a model of \mathcal{P} . We call J the result of applying \mathcal{P} to I , and denote J by $\mathcal{P}(I)$.

Stratified programs are essentially sequences of semipositive programs. Formally, a program \mathcal{P} is called *stratified* if it can be partitioned into parts $\mathcal{P}_1, \dots, \mathcal{P}_n$ called *strata*, such that **(i)** the strata have pairwise disjoint sets of intensional shape names; **(ii)** each stratum is semipositive; and **(iii)** the strata are ordered in such a way that when a shape name s occurs in the body of a rule in some stratum, s is not intensional in any later stratum.

Let \mathcal{P} be a stratified program with n strata $\mathcal{P}_1, \dots, \mathcal{P}_n$ and let again I be an interpretation over a vocabulary without the intensional shape names. We define $\mathcal{P}(I)$, the result of applying \mathcal{P} to I , to be the interpretation J_n , where $J_0 := I$ and $J_{k+1} := \mathcal{P}_{k+1}(J_k)$ for $0 \leq k < n$.

Stratified shape schemas

We are now ready to define a *stratified shape schema* again as a set of inclusions, but now paired with a stratified program. Formally, it is a pair $(\mathcal{P}, \mathcal{T})$, where:

- \mathcal{P} is a program that is stratified, and where every shape name mentioned in the body of some rule is an intensional shape name in \mathcal{P} .
- \mathcal{T} is a finite set of inclusion statements $\phi_1 \subseteq \phi_2$, where ϕ_1 and ϕ_2 mention only shape names that are intensional in \mathcal{P} .

15:14 Expressiveness of SHACL Features

Now we define a graph G to *conform* to $(\mathcal{P}, \mathcal{T})$ if $\llbracket \phi_1 \rrbracket^{\mathcal{P}(G)}$ is a subset of $\llbracket \phi_2 \rrbracket^{\mathcal{P}(G)}$, for every inclusion $\phi_1 \subseteq \phi_2$ in \mathcal{T} .

► Remark 18. The nonrecursive notion of shape schema, defined in Section 2, corresponds to the special case where \mathcal{P} is the empty program.

Extending Theorem 4

Theorem 4 extends to stratified shape schemas. Indeed, consider a stratified shape schema $(\mathcal{P}, \mathcal{T})$. Shapes not mentioning any shape names are referred to as *elementary shapes*. We observe that for every intensional shape name s and every graph H , there exists an elementary shape ϕ such that $\llbracket s \rrbracket^{\mathcal{P}(H)} = \llbracket \phi \rrbracket^H$. Furthermore, ϕ uses the same constants, quantifiers, and path expressions as \mathcal{P} . For semipositive programs, this is shown using a fixpoint characterization of the minimal model; for stratified programs, this argument can then be applied repeatedly. The crux, however, is that graphs G and G' of Proposition 5 will have the same ϕ . Indeed, by that Proposition, the fixpoints of the different strata will be reached on G and on G' in the same stage. We effectively obtain an extension of Proposition 5, which establishes the theorem for features X other than *closed*.

Also for $X = \textit{closed}$, the reasoning, given after Lemma 12, extends in the same way to stratified shape schemas, since the graphs G and G' used there again yield exactly the same evaluation for all shapes that do not use *closed*.

Extending Theorem 14

Also Theorem 14 extends to stratified shape schemas. Thereto, Lemma 12 needs to be reproven in the presence of a stratified program \mathcal{P} defining the intensional shape names. The extended Lemma 12 then states that $\llbracket \phi \rrbracket^{\mathcal{P}(G)} = \llbracket \phi \rrbracket^{\mathcal{P}(G')}$. The proof of Theorem 14 then goes through unchanged.

5 Concluding remarks

An obvious open question is whether our results extend further to nonstratified programs, depending on various semantics that have been proposed for Datalog with negation, notably well-founded or stable models [1, 20]. One must then deal with 3-valued models and, for stable models, choose whether the TBox should hold in every stable model (skeptical), or in at least one (credulous). For example, Andreşel et al. [2] adopt a credulous approach. In the same vein, even for stratified programs, one may consider *maximal* models instead of minimal ones, as suggested for ShEx [6].

Notably, Corman et al. [9] have already suggested that disjointness is redundant in a setting of recursive shape schemas with nonstratified negation. Their expression is not correct, however [17].⁴

A quirk in the design of SHACL is that it only allows equality and disjointness tests $eq(E_1, E_2)$ and $disj(E_1, E_2)$ where E_1 can be a general path expression, but E_2 needs to be a property name. It is open whether allowing “full” equality or disjointness tests, i.e., allowing a general path expression for E_2 , would strictly increase expressive power. Furthermore,

⁴ Their approach is to postulate two shape names s_1 and s_2 that can be assigned arbitrary sets of nodes, as long as the two sets form a partition of the domain. Then for one node x to satisfy the shape $disj(E, p)$, it is sufficient that $E(x)$ is a subset of s_1 and $p(x)$ of s_2 . This condition is not necessary, however, as other nodes may require different partitions.

for $X = \text{disj}$, our proof of Proposition 5, and thus Theorem 4, no longer works under full equality tests. For X one of the other three features, however, the proof continues to work under full disjointness or equality tests.

Not included in our formal syntax, but allowed in SHACL, are shapes of the form $\text{eq}(id, p)$ and $\text{disj}(id, p)$, where id stands for the focus node. These are allowed in SHACL under “node shapes”. In our formalism, $\text{eq}(id, p)$ is already expressible as $\text{eq}(p?, p) \wedge \geq_1 p. \top \wedge \leq_1 p. \top$. Furthermore, $\text{disj}(id, p)$ is expressible as $\neg \text{eq}(p?, p)$. Note, however, the use of $?$ in these expressions. Hence, when investigating the primitivity of $?$, as we have done in this paper, it would be more proper to include $\text{eq}(id, p)$ and $\text{disj}(id, p)$ directly in the syntax. Whether $?$ remains primitive in this setting is left as an open problem.

Another open problem, also concerning the primitivity of $?$, is to consider the more restrictive setting where, in each shape of the form $\geq_n E.\phi$, n equals 1 or 2. Indeed, the higher counts seem rarely used in practice. Since our query showing primitivity of $?$ involves counting to 4, it remains open whether $?$ still adds expressive power when counts are limited to 1 or 2. (We thank an anonymous reviewer for suggesting this problem.)

Finally, a general question surrounding SHACL, even standard nonrecursive SHACL, is to understand better in which sense (if at all) this language is actually better suited for expressing constraints on RDF graphs than, say, SPARQL ASK queries [8, 19, 10]. Certainly, the affinity with description logics makes it easy to carve out cases where higher reasoning tasks become decidable [13, 15]. It is also possible to show that nonrecursive SHACL is strictly weaker in expressive power than SPARQL. But does SHACL conformance checking really have a lower computational complexity? Can we think of novel query processing strategies that apply to SHACL but not easily to SPARQL? Are SHACL expressions typically shorter, or perhaps longer, than the equivalent SPARQL ASK expression? How do the expression complexities [21] compare?

References

- 1 S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.
- 2 M. Andreşel, J. Corman, M. Ortiz, J.L. Reutter, O. Savkovic, and M. Simkus. Stable model semantics for recursive SHACL. In Y. Huang, I. King, T.-Y. Liu, and M. van Steen, editors, *Proceedings WWW'20*, pages 1570–1580. ACM, 2020.
- 3 A.K. Aylamazyan, M.M. Gilula, A.P. Stolboushkin, and G.F. Schwartz. Reduction of the relational model with infinite domains to the case of finite domains. *Doklady Akademii Nauk SSSR*, 286(2):308–311, 1986. In Russian.
- 4 F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.
- 5 B. Bogaerts, M. Jakubowski, and J. Van den Bussche. SHACL: A description logic in disguise. *CoRR*, abs/2108.06096, 2021. URL: <https://arxiv.org/abs/2108.06096>.
- 6 I. Boneva, J.E.L. Gayo, and E.G. Prud'hommeaux. Semantics and validation of shape schemas for RDF. In C. d'Amato, M. Fernandez, V. Tamma, et al., editors, *Proceedings 16th International Semantic Web Conference*, volume 10587 of *Lecture Notes in Computer Science*, pages 104–120. Springer, 2017.
- 7 D. Calvanese, G. De Giacomo, D. Nardi, and M. Lenzerini. Reasoning in expressive description logics. In F. Baader, D. Calvanese, D. McGuinness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook*, chapter 23. Cambridge University Press, 2003.
- 8 J. Corman, F. Florenzano, J.L. Reutter, and O. Savkovic. Validating SHACL constraints over a SPARQL endpoint. In C. Ghidini, O. Hartig, M. Maleshkova, V. Svátek, et al., editors, *Proceedings 18th International Semantic Web Conference*, volume 11778 of *Lecture Notes in Computer Science*, pages 145–163. Springer, 2019.

- 9 J. Corman, J.L. Reutter, and O. Savkovic. Semantics and validation of recursive SHACL. In D. Vrandeć et al., editors, *Proceedings 17th International Semantic Web Conference*, volume 11136 of *Lecture Notes in Computer Science*, pages 318–336. Springer, 2018. Extended version, technical report KRDB18-01, <https://www.inf.unibz.it/krdp/tech-reports/>.
- 10 B. De Meester, P. Heyvaert, et al. RDF graph validation using rule-based reasoning. *Semantic Web*, 12(1):117–142, 2021.
- 11 I. Horrocks, O. Kutz, and U. Sattler. The even more irresistible SROIQ. In P. Doherty, J. Mylopoulos, and C.A. Welty, editors, *Proceedings 10th International Conference on Principles of Knowledge Representation and Reasoning*, pages 57–67. AAAI Press, 2006.
- 12 R. Hull and J. Su. Domain independence and the relational calculus. *Acta Informatica*, 31:513–524, 1994.
- 13 M. Leinberger, P. Seifer, et al. Deciding SHACL shape containment through description logics reasoning. In J.Z. Pan et al., editors, *Proceedings 19th International Semantic Web Conference*, volume 12506 of *Lecture Notes in Computer Science*, pages 366–383. Springer, 2020.
- 14 OWL 2 Web ontology language: Structural specification and functional-style syntax. W3C Recommendation, December 2012.
- 15 P. Pareti, G. Konstantinidis, et al. SHACL satisfiability and containment. In J.Z. Pan et al., editors, *Proceedings 19th International Semantic Web Conference*, volume 12506 of *Lecture Notes in Computer Science*, pages 474–493. Springer, 2020.
- 16 RDF 1.1 primer. W3C Working Group Note, June 2014.
- 17 J. Reutter, January 2021. Personal communication.
- 18 Shapes constraint language (SHACL). W3C Recommendation, July 2017.
- 19 J. Tao et al. Integrity constraints in OWL. In *Proceedings 24th AAAI Conference on Artificial Intelligence*, pages 1443–1448, 2010.
- 20 M. Truszczyński. An introduction to the stable and well-founded semantics of logic programs. In M. Kifer and Y.A. Liu, editors, *Declarative Logic Programming: Theory, Systems, and Applications*, pages 121–177. ACM and Morgan & Claypool, 2018.
- 21 M. Vardi. The complexity of relational query languages. In *Proceedings 14th ACM Symposium on the Theory of Computing*, pages 137–146, 1982.