EXPRESSIVENESS OF SHACL FEATURES AND EXTENSIONS FOR FULL EQUALITY AND DISJOINTNESS TESTS

DOI: 10.46298/LMCS-20(1:16)2024
Handle: http://hdl.handle.net/1942/42802

Logical Methods in Computer Science
Volume 20, Issue 1, 2024, pp. 16:1–16:34
https://lmcs.episciences.org/

Submitted Dec. 08, 2022
Published Feb. 21, 2024

EXPRESSIVENESS OF SHACL FEATURES AND EXTENSIONS FOR FULL EQUALITY AND DISJOINTNESS TESTS

BART BOGAERTS [a], MAXIME JAKUBOWSKI [a,b], AND JAN VAN DEN BUSSCHE [b]

[a] Vrije Universiteit Brussel, Belgium
e-mail address: bart.bogaerts@vub.be

[b] Universiteit Hasselt, Belgium
e-mail address: maxime.jakubowski@uhasselt.be
e-mail address: jan.vandenbussche@uhasselt.be

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 three fundamental features that are not so common in description logics. These features are 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 three 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.

In addition, we show that enriching SHACL with "full" versions of equality tests, or disjointness tests, results in a strictly more powerful language.

1. Introduction

On the Web, the Resource Description Framework (RDF [RDF14]) 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 [BGP17] and SHACL [SHA17]. 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

Key words and phrases: Expressive power, schema languages.
This paper is an extended version of our work presented at ICDT 2022 [BJVdB22].

DOI:10.46298/LMCS-20(1:16)2024

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) "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 [CRS18], and used in subsequent works by several authors [ACO+20, LS+20, PK+20]. That formalization reveals a striking similarity between shapes on the one hand, and concepts, familiar from description logics [BHLS17], 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 $(\mathsf{author}^-/\mathsf{author})^*$. Here, the minus superscript denotes an inverse edge.

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

[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.

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

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.

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 [CDGNL03]: 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) $\geq_1 \mathsf{phone}.\top \wedge \neg \geq_1 \mathsf{email}.\top$. This uses qualified number restriction with count 1 (so essentially existential quantification), conjunction, and negation; $\top$ stands for true.
(2) $\geq_5 \mathsf{managed\text{-}by}^-.\top$. This uses counting to 5, and inverse.
(3) $\geq_1 \mathsf{friend}^*/\mathsf{CEO\text{-}of}.\{\mathsf{Apple}\}$. This uses a regular path expression and the constant Apple.

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

Equality: 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.

Disjointness: A similar shape $disj(E, r)$ tests disjointness of the two sets of reachable nodes. To illustrate, example shape 4 is expressed as $\neg disj(\mathsf{colleague}, \mathsf{friend})$.

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 $R = \{\mathsf{name}, \mathsf{address}, \mathsf{birthdate}\}$.

Our goal in this paper is to clarify the impact of the above three 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 three 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.
- We additionally show that "full" variants of equality tests or disjointness tests result in strictly more expressive languages. This result anticipates planned extensions of SHACL [Knu21, Jak22].
- 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, Corman, et al. cited above. Section 3 and Section 4 present our results, and Section 5 extends our result for "full" equality and disjointness tests. Section 6 presents the extension to stratified recursion. Section 7 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 6 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:

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

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*. In p^-, $-$ stands for inverse.

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

Remark 2.1. 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 2.2. 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 $\neg \geq_{n+1} E.\phi$.

Remark 2.3. In our formalization, a path expression can be 'id'. We show in Lemma 3.3 that every path expression is equivalent to id, $E' \cup id$ or E', where E' does not use id. In real SHACL, it is possible to write $E' \cup id$ using "zero-or-one" path expressions. Explicitly writing id is not possible, but this poses no problem. Path expressions can only appear in counting quantifiers, equality and disjointness shapes. The shape $\geq_n id.\phi$ is clearly equivalent to ϕ if $n = 1$, otherwise, it is equivalent to $\neg\top$. The shapes $eq(E,p)$ or $disj(E,p)$ where E is id are implicitly expressible in SHACL by writing the equality or disjointness constraint in node shapes, rather than property shapes.

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

[3] In practice, node names and property names are IRIs [RDF14], 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.

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

Table 1: Semantics of path expressions.

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

Table 2: Conditions for conformance of a node to a shape.

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

The semantics of shapes is now defined as follows.

- On any interpretation I as above, every path expression E over Σ evaluates to a binary relation $[\![E]\!]^I$ 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 $I, a \vDash \phi$. 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 $[\![E]\!]^I$.
  - We also use the notation $\sharp X$ for the cardinality of a set X.
- For a shape ϕ and interpretation I, the notation
$$[\![\phi]\!]^I := \{a \in \Delta^I \mid I, a \vDash \phi\}$$
will be convenient.

Example 2.4. In the Introduction we already gave three examples (1), (2), and (3) of shapes expressed in English and the formal syntax. The target query of example (7) from the Introduction can be expressed as the shape $\exists \mathsf{email}^-.\top$. The shape of example (7) can be written as $\leq_1 \mathsf{email}^-.\top$.

2.2. **Graphs and their interpretation.** Remember from Table 2 that a shape $closed(R)$ states that the focus node may only have outgoing properties that are mentioned in R. It may appear that such a shape 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,

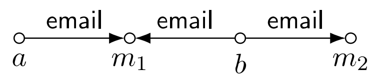

Figure 1: An example graph G_{ex}

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).
- $[\![c]\!]^G$ equals c itself, for every node name c.
- $[\![p]\!]^G$ equals the set of p-edges in G, for every property name p.

Note that since graphs are finite, $[\![p]\!]^G$ 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 $[\![E]\!]^G$ on N is well-defined; for any shape ϕ and $a \in N$, it is well-defined whether or not $G, a \vDash \phi$; and we can also use the notation $[\![\phi]\!]^G$.

Remark 2.5. 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 [AHV95, Theorem 5.6.1], [AGSS86, HS94]. 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;
- $[\![c]\!]^I = c$ for each node name $c \in \Sigma$;
- $[\![p]\!]^I = [\![p]\!]^G$ 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 $x \in [\![\phi]\!]^G$ if and only if $x \in [\![\phi]\!]^I$. For all other node names x, one can show that $x \in [\![\phi]\!]^G$ if and only if $\star \in [\![\phi]\!]^I$.

Example 2.6 (Example 2.4 continued)**.** Consider the graph G_{ex} depicted in Figure 1. This graph can be seen as the interpretation I_{ex} with an infinite domain containing the elements a, b, m_1, and m_2. It interprets the predicate name email as $\{(a, m_1), (b, m_1), (b, m_2)\}$ and all other predicate names as the empty set. If we look at the interpretation of example (7) from the Introduction in I_{ex}, we have $[\![\leq_1 \mathsf{email}^-.\top]\!]^{I_{ex}} = \{m_1\}$ for the shape, and $[\![\exists \mathsf{email}^-.\top]\!]^{I_{ex}} = \{m_1, m_2\}$ for the target.

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: $\exists \mathsf{type}/\mathsf{subclass}^*.\{c\}$ for any constant c. Here, type and subclass represent distinguished IRIs from the RDF Schema vocabulary [RDF14].

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 $G \vDash \mathcal{S}$, if $[\![\phi_1]\!]^G$ is a subset of $[\![\phi_2]\!]^G$, 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

$$[\![\mathcal{S}]\!] := \{\text{graph } G \mid G \vDash \mathcal{S}\}.$$

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

Example 2.7 (Example 2.6 continued)**.** Constraint (7) from the Introduction can be written as:

$$\exists \mathsf{email}^-.\top \subseteq\ \leq_1 \mathsf{email}.\top$$

We can see that G_{ex} does not satisfy the constraint, because $\{m_1, m_2\} \not\subseteq \{m_2\}$. However, if we remove the triple (b, email, m_1) from G_{ex}, then the shape is interpreted as $[\![\leq_1 \mathsf{email}.\top]\!]^{I_{ex}} = \{m_1, m_2\}$ and the constraint does hold, as $\{m_1, m_2\} \subseteq \{m_1, m_2\}$.

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*, and *closed*. 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 three features. Specifically, they only involve shapes built from boolean connectives, constants, and qualified number restrictions, with path expressions built from property names, *id* and 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,

$$F_1 \preceq F_2 \quad \Leftrightarrow \quad F_1 \subseteq F_2, \tag{$*$}$$

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 $\exists r.\top \subseteq closed(r)$.

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

Theorem 3.1. *Let $X \in \{eq, disj, closed\}$ and let F be a feature set with $X \notin F$. Then Q_X is not definable in $\mathcal{L}(F)$.*

For $X = closed$, Theorem 3.1 is proven differently than for the other two features. First, we deal with the remaining features through the following concrete result, illustrated in Figure 2. The formal definition of the graphs illustrated in Figure 2 for $X = disj$ will be provided in Definition 3.8.

Proposition 3.2. *Let $X = disj$ or eq. 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*
$$[\![\phi]\!]^G = [\![\phi]\!]^{G'}.$$

Here, "counting to at most m" means that all quantifiers $\geq_n$ used in ϕ satisfy $n \leq m$. For $X = eq$, this proposition is reformulated as Proposition 3.12, and for $X = disj$, this proposition is reformulated as Proposition 3.15.

To see that Proposition 3.2 indeed establishes Theorem 3.1 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 of the "validation report" in SHACL [SHA17]: 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.

We will prove Proposition 3.2 for $X = disj$ in Section 3.2, and $X = eq$ in Section 3.3. We will show Theorem 3.1 for $X = closed$ in Section 3.4. However, we first need to establish some preliminaries on path expressions.

3.1. Preliminaries on path expressions.

We call a path expression E *equivalent* to a path expression E' when for every graph G, $[\![E]\!]^G = [\![E']\!]^G$. We call a path expression E *id-free* whenever *id* is not present in the expression.

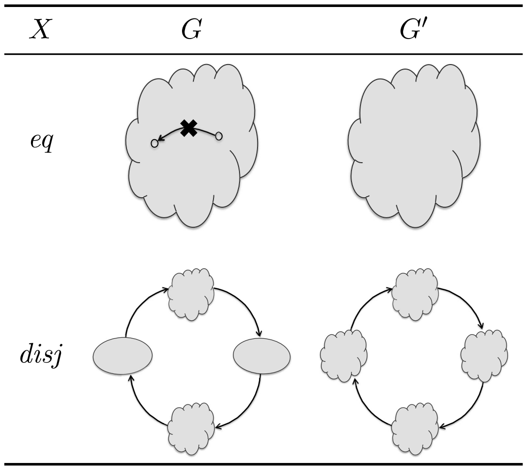

Figure 2: Graphs used to prove Proposition 3.2. The nodes are taken outside Σ. For $X = eq$, the cloud shown for G' represents a complete directed graph on $m+1$ nodes, with self-loops, and G is the same graph with one directed edge removed. For $X = disj$, in the picture for G, each cloud again stands 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.

Lemma 3.3. *Every path expression E is equivalent to: id, or $E' \cup id$, or E' where E' is an id-free path expression.*

Proof. The proof is by induction on the structure of E. When E is id-free or id, the claim directly follows. We consider the following inductive cases:

- E is E_1/E_2. By induction, we consider nine cases. When both E_1 and E_2 are id-free, E is id-free. Whenever E_1 is id, clearly E is equivalent to E_2. Analogously, whenever E_2 is id, E is equivalent to E_1.

  Consider the two cases where E_1 is $E_1' \cup id$ with E_1' an id-free path expression. First, when E_2 is $E_2' \cup id$ with E_2' an id-free path expression, then E is equivalent to $E_1'/E_2' \cup E_1' \cup E_2' \cup id$ which is of the form $E' \cup id$ with E' the id-free path expression $E_1'/E_2' \cup E_1' \cup E_2'$. Second, when E_2 is id-free, E is equivalent to $E_1'/E_2 \cup E_2$ which is id-free.

  Finally, consider the two case where E_1 is id-free and E_2 is $E_2' \cup id$ with E_2' an id-free path expression, then E is equivalent to $E_1/E_2' \cup E_1$ which is id-free.
- E is $E_1 \cup E_2$. This case follows immediately by induction.
- E is E_1^*. There are three cases. When E_1 is id, E is equivalent to id. When E_1 is $E_1' \cup id$ with E_1' an id-free path expression, then E is equivalent to $E_1'^*$ and clearly $E_1'^*$ is id-free. Lastly, when E_1 is id-free, clearly E is as well. □

We also need the notion of "safe" path expressions together with the following Lemma, detailing how path expressions can behave on the nodes outside a graph. One can divide all path expressions into the "safe" and the "unsafe" ones.

Definition 3.4 (Safety)**.** A path expression is *safe* if one of the following conditions holds:

- E is p or p^- with p a property name
- E is $E_1 \cup E_2$ and both E_1 and E_2 are safe
- E is E_1/E_2 and at least one of E_1 or E_2 is safe

Otherwise, E is *unsafe*.

Lemma 3.5. *Let E be an id-free path expression and let G be a graph.*
- *If E is safe, then $[\![E]\!]^G \subseteq N_G \times N_G$.*
- *If E is unsafe, then $[\![E]\!]^G = ([\![E]\!]^G \cap N_G \times N_G) \cup \{(a,a) \mid a \in N - N_G\}$.*

Proof. By induction. If E is a property name or its inverse, then the claim clearly holds. Now assume E is of the form $E_1 \cup E_2$. The cases where both E_1 and E_2 are safe, or both are unsafe, are clear by induction. If E_1 is safe but E_2 is not, then $[\![E]\!]^G = [\![E_1]\!]^G \cup [\![E_2]\!]^G = ([\![E_1]\!]^G \cap N_G \times N_G) \cup ([\![E_2]\!]^G \cap N_G \times N_G) \cup \{(a,a) \mid a \in N - N_G\} = [\![E]\!]^G \cap (N_G \times N_G) \cup \{(a,a) \mid a \in N - N_G\}$. The same reasoning can be used when E_2 is safe but E_1 is not.

Next, assume E is of the form E_1/E_2. Furthermore assume E_1 is safe, so that E is safe. Let $(x,y) \in [\![E]\!]^G$. Then there exists z such that $(x,z) \in [\![E_1]\!]^G$ and $(z,y) \in [\![E_2]\!]^G$. Since E_1 is safe, x and z are in N_G. Now regardless of whether E_2 is safe or not, since $(z,y) \in [\![E_2]\!]^G$ and $z \in N_G$, we get $y \in N_G$ as desired. The same reasoning can be used when E_2 is safe.

If E is not safe, we verify that $[\![E]\!]^G = ([\![E]\!]^G \cap N_G \times N_G) \cup \{(a,a) \mid a \in N - N_G\}$. For the inclusion from left to right, take $(x,y) \in [\![E]\!]^G$. Then there exists z such that $(x,z) \in [\![E_1]\!]^G$ and $(z,y) \in [\![E_2]\!]^G$. By induction, there are four cases. If both (x,z) and (z,y) are in $N_G \times N_G$, then clearly $(x,y) \in [\![E]\!]^G \cap N_G \times N_G$. If both $(x,z),(z,y)$ are in $\{(a,a) \mid a \in N - N_G\}$ clearly $(x,y) \in \{(a,a) \mid a \in N - N_G\}$. Lastly, the two cases where one of (x,z) and (z,y) is in $N_G \times N_G$ and the other in $\{(a,a) \mid a \in N - N_G\}$, are not possible.

For the inclusion from right to left, take $(x,y) \in [\![E]\!]^G \cap (N_G \times N_G) \cup \{(a,a) \mid a \in N - N_G\}$. If $(x,y) \in [\![E]\!]^G \cap N_G \times N_G$ then $(x,y) \in [\![E]\!]^G$. Otherwise, $(x,y) = (a,a)$ for some $a \in N - N_G$. Then $(a,a) \in [\![E_1]\!]^G$ and $(a,a) \in [\![E_2]\!]^G$ since E_1 and E_2 are not safe. We conclude $(a,a) \in [\![E_1/E_2]\!]^G$ as desired.

Next, assume E is of the form E_1^*. Note that E is unsafe. By definition of Kleene star, we only need to verify that $[\![E]\!]^G \subseteq ([\![E]\!]^G \cap N_G \times N_G) \cup \{(a,a) \mid a \in N - N_G\}$. Let $(x,y) \in [\![E]\!]^G$. If $x = y$, the claim clearly holds. Otherwise, we consider two cases:
- If E_1 is safe, we know $[\![E_1]\!]^G \subseteq N_G \times N_G$. Clearly the reflexive-transitive closure of a subset of $N_G \times N_G$ is also a subset of $N_G \times N_G$. Therefore, $(x,y) \in N_G \times N_G$ as desired.
- If E_1 is unsafe, then by induction $[\![E_1]\!]^G = ([\![E_1]\!]^G \cap N_G \times N_G) \cup \{(a,a) \mid a \in N - N_G\}$. As $x \neq y$ we know (x,y) is in the reflexive-transitive closure of $[\![E_1]\!]^G \cap N_G \times N_G$ which is a subset of $N_G \times N_G$. □

Lastly, we define the notion of a *string*, together with the following Lemma, detailing a convenient property of path expressions.

Definition 3.6. A string s is a path expression of the form: id, or s'/p or s'/p^- where s' is a string and p is a property name.

Lemma 3.7. *For every path expression E and every natural number n, there exists a finite non-empty set of strings U s.t. for every graph G with at most n nodes we have $[\![E]\!]^G = \bigcup_{s \in U} [\![s]\!]^G$.*

The proof of Lemma 3.7 can be found in the appendix.

3.2. **Disjointness.** 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 proof for $X = eq$.

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

Definition 3.8 ($G_{disj}(\Sigma, m)$)**.** 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 $j = 1, \ldots, M$. (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. We describe these edges next. 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 $j \neq j'$. So, formally, we have: $G_{disj}(\Sigma, m) := \{(x_i^j, p, x_{i \bmod 4+1}^{j'}) \mid i \in \{1, \ldots, 4\}$ and $j, j' \in \{1, \ldots, M\}$ and $p \in \Sigma \cap P\} \cup \{(x_i^j, p, x_i^{j'}) \mid i \in \{1, \ldots, 4\}$ and $j, j' \in \{1, \ldots, M\}$ and $j \neq j'$ and $p \in \Sigma \cap P\}$.

Thus, in Figure 2, 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 3.9 ($G'_{disj}(\Sigma, m)$)**.** 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 even i values).

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

Lemma 3.10. *Let G be $G_{disj}(\Sigma, m)$. Call a path expression* simple *if it is a union of expressions of the form $s_1/\ldots/s_n$, where $n \geq 1$ and one of the s_i is a property name while the other s_i are "id". Let E be a non-simple, id-free path expression over Σ. The following three statements hold:*

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

Proof. For $i = 1, 2, 3, 4$, define the *i-th blob of nodes* to be the set $X_i = \{x_i^1, \ldots, x_i^M\}$ (see Figure 2). 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 $[\![p^-]\!]^G(v) \supseteq X_{prev(i)}$ and clearly $X_{prev(i)} - [\![r]\!]^G(v) \neq \emptyset$. 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 $[\![E]\!]^G \supseteq [\![E_1]\!]^G$ and $[\![E]\!]^G \supseteq [\![E_2]\!]^G$. 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^*. If E_1 is not simple, the three statements follow immediately by induction, since $[\![E]\!]^G \supseteq [\![E_1]\!]^G$. 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 $[\![E]\!]^G(v)$, but not in $[\![r]\!]^G(v)$, 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:
  - $[\![E]\!]^G(v) \supseteq X_i$. Let $u \in X_i$. If $u \neq v$, choose a third node $w \in X_i$. Since X_i is a clique, $(v, w) \in [\![E_1]\!]^G$ regardless of whether case A or B applies to E_1. By case A for E_2, we also have $(w, u) \in [\![E_2]\!]^G$, whence $u \in [\![E]\!]^G(v)$ as desired. If $u = v$, we similarly have $(v, w) \in [\![E_1]\!]^G$ and $(w, u) \in [\![E_2]\!]^G$ as desired.
  - $[\![E]\!]^G(v) \supseteq X_{next(i)}$. 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 $(v, w) \in [\![E_1]\!]^G$. By case A for E_2, we also have $(w, u) \in [\![E_2]\!]^G$, whence $u \in [\![E]\!]^G(v)$ as desired.

  We conclude that $[\![E]\!]^G(v) \supseteq X_i \cup X_{next(i)} \supseteq [\![r]\!]^G$ 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 $[\![E]\!]^G(v) \supseteq X_i \cup X_{prev(i)}$.

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

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 $[\![r]\!]^G(v) = X_{next(i)}$. To verify that $[\![E]\!]^G(v) \supseteq X_{next(i)}$, 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 $(v, w) \in [\![E_1]\!]^G$. Moreover, since $X_{next(i)}$ is a clique, $(w, u) \in [\![E_2]\!]^G$ regardless of whether case A or B applies to E_2. We obtain $(v, u) \in [\![E]\!]^G$ 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 $[\![E]\!]^G(v)$, and clearly these u are not in $X_{next(i)} = [\![r]\!]^G(v)$.
  - If case B applies to E_2, then, since X_i is a clique, any node $u \in X_i$ belongs to $[\![E]\!]^G(v)$, 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 $[\![E]\!]^G(v) \supseteq X_{prev(i)}$. 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.

Lemma 3.11. *Let G' be $G'_{disj}(\Sigma, m)$ and let E be a non-simple, id-free path expression over Σ. The following statements hold:*

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

Proof. The proof is similar to the proof of Lemma 3.10, but simpler due to the homogeneous nature of the graph G'. We omit the proof. □

We are now ready to prove the non-obvious part of Proposition 3.2 where $X = disj$. We use the following version of the proposition.

Proposition 3.12. *Let V be the common set of nodes of the graphs $G = G_{disj}(\Sigma, m)$ and $G' = G'_{disj}(\Sigma, m)$. Let ϕ be a shape over Σ that does not use disj, and that counts to at most m. Then either $[\![\phi]\!]^G \cap V = \emptyset$ or $[\![\phi]\!]^G \supseteq V$. Moreover, $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$.*

Proof. This is proven by induction on the structure of ϕ. Let H be G or G'. If ϕ is $\top$, then $[\![\top]\!]^H = N \supseteq V$. If ϕ is $\{c\}$, then $[\![\{c\}]\!]^H = \{c\} \subseteq \Sigma$ and we know that $\Sigma \cap V = \emptyset$. Next assume ϕ is of the form $eq(E, p)$. Using Lemma 3.3, we distinguish four different cases for E.

- E is id. According to Lemma 3.10 and Lemma 3.11 $[\![E]\!]^H$ will always contain either $[\![p]\!]^H$ or $[\![p^-]\!]^H$. In both cases, $[\![E]\!]^H(v)$ clearly never equals $[\![id]\!]^H(v) = \{v\}$. Therefore, $[\![\phi]\!]^H \cap V = \emptyset$.
- E is $E' \cup id$ where E' is id-free or E itself is id-free and non-simple. Lemmas 3.10 and 3.11 tell us that $[\![E]\!]^H(v) - [\![r]\!]^H(v) \neq \emptyset$ for every $v \in V$. Since $[\![r]\!]^H = [\![p]\!]^H$, this means $H, v \nvDash \phi$ for $v \in V$, or, equivalently, $[\![\phi]\!]^G \cap V = \emptyset$. To see that, moreover, $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$, it remains to show that $G, v \vDash \phi$ iff $G', v \vDash \phi$ for all node names $v \notin V$.
- E is id-free and simple. Then $[\![E]\!]^H = [\![p]\!]^H$, so clearly $[\![\phi]\!]^H = N \supseteq V$.

We still need to show $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$. Clearly, $[\![p]\!]^G(v) = [\![p]\!]^{G'}(v) = \emptyset$. Now by Lemma 3.5, if E is safe, then also $[\![E]\!]^G(v) = [\![E]\!]^{G'}(v) = \emptyset$, so $G, v \vDash \phi$ and $G', v \vDash \phi$. On the other hand, if E is unsafe, then by the same Lemma $[\![E]\!]^G(v) = [\![E]\!]^{G'}(v) = \{v\} \neq \emptyset$, so $G, v \nvDash \phi$ and $G', v \nvDash \phi$, as desired.

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

We next consider the inductive cases. The cases for the boolean connectives follow readily by induction. Finally, assume ϕ is of the form $\geq_n E.\phi_1$. By induction, there are two possibilities for ϕ_1:

- If $[\![\phi_1]\!]^H \cap V = \emptyset$, then also $[\![\phi]\!]^H \cap V = \emptyset$ since path expressions can only reach nodes in some graph from nodes in that graph.
- If $[\![\phi_1]\!]^H \supseteq V$, we distinguish three cases using Lemma 3.3. First, when E is id, then if $n = 1$, $[\![\phi]\!]^H \supseteq V$. Otherwise, if $n \neq 1$, then $[\![\phi]\!]^H = \emptyset$. Next, when E is id-free or $E' \cup id$ with E' an id-free path expression, it suffices to show that $\sharp[\![E']\!]^H(v) \geq n$ for all $v \in V$. By Lemmas 3.10 and 3.11 we know that $[\![E_1]\!]^H(v)$ contains $[\![r]\!]^H(v)$ or $[\![r^-]\!]^H(v)$. Inspecting H, we see that each of these sets has at least $\max(3, m) \geq n$ elements, as desired. Finally, when E is equivalent to an id-free path expression or whenever E simply does not use id, the argument is analogous to the previous case.

In both cases we still need to show that $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$. We already showed that $[\![\phi]\!]^G \supseteq V$ and $[\![\phi]\!]^{G'} \supseteq V$, or $[\![\phi]\!]^G \cap V = \emptyset$ and $[\![\phi]\!]^{G'} \cap V = \emptyset$. 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 $x \in [\![\phi]\!]^G - V$. Since $G, x \vDash \phi$, there exist y_1, $\dots$, y_n such that $(x, y_i) \in [\![E]\!]^G$ and $G, y_i \vDash \phi_1$ for $i = 1, \dots, n$. However, since $x \notin V$, by

Lemma 3.5, all y_i must equal x. Hence, $n = 1$ and $(x, x) \in [\![E]\!]^G$ and $G, x \vDash \phi_1$. Then again by the same Lemma, $(x, x) \in [\![E]\!]^{G'}$, since G and G' have the same set of nodes V. Moreover, by induction, $G', x \vDash \phi_1$. We conclude that $G', x \vDash \phi$ as desired. The inclusion from right to left is argued symmetrically. □

3.3. **Equality.** Next, we turn our attention to Proposition 3.2 for $X = eq$. We define the graphs from Figure 2 formally.

Definition 3.13. Let Σ be a finite vocabulary including r, and let m be a natural number. Choose a set V of node names outside Σ, of cardinality $M := \max(3, m + 1)$. Fix two arbitrary nodes a and b from V. We define the graph $G_{eq}(\Sigma)$ over the set of property names from Σ as follows. For each property name p in Σ, the set of p-edges in $G_{eq}(\Sigma)$ equals $V \times V - (b, a)$. We define the graph $G'_{eq}(\Sigma)$ similarly, but with $V \times V$ as the set of p-edges.

So, $G'_{eq}(\Sigma, m)$ is a complete graph, and $G_{eq}(\Sigma, m)$ is a complete graph with one edge (b, a) removed.

Lemma 3.14. *Let E be an id-free path expression over Σ and let $H = G_{eq}(\Sigma, m)$ or $G'_{eq}(\Sigma, m)$. Then*

A. $[\![E]\!]^H \supseteq [\![r]\!]^H$, *or*
B. $[\![E]\!]^H \supseteq [\![r^-]\!]^H$.

Proof. The claim is obvious for $G'_{eq}(\Sigma, m)$, being a complete graph. So we focus on the graph $G_{eq}(\Sigma, m)$. The proof is by induction. If E is a property name or its inverse, the claim is clear. If E is of the form $E_1 \cup E_2$, the claim is immediate by induction.

Assume E is of the form E_1/E_2. We show that A applies.[4] If A applies to E_1, this is clear, since we can follow any edge by E_1 and then stay at the head of the edge by E_2 using the self-loop. If B applies to E_1, the same can still be done for all edges except for (a, b), which is the only nonsymmetrical edge. To go from a to b by E, we go by E_1 from a to a node c distinct from a and b, then go by E_2 from c to b.

If E is of the form E_1^*, again A applies, since E_1^* contains E_1/E_1. □

We are now ready to prove the non-obvious part of Proposition 3.2 where $X = eq$. We use the following version of the proposition.

Proposition 3.15. *Let G be $G_{eq}(\Sigma, m)$ and let G' be $G'_{eq}(\Sigma, m)$. Let ϕ be a shape over Σ that does not use eq and that counts to at most m. Then either $[\![\phi]\!]^G \cap V = \emptyset$ or $[\![\phi]\!]^G \supseteq V$. Moreover, $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$.*

Proof. This is proven by induction on the structure of ϕ. Let H be G or G'. We focus directly on the relevant cases. Assume ϕ is of the form $disj(E_1, E_2)$. Lemma 3.14 clearly yields that $[\![\phi]\!]^H \cap V = \emptyset$. It again remains to verify that $G, v \vDash \phi$ iff $G', v \vDash \phi$ for all node names $v \notin V$. By Lemma 3.5, for such v and $H = G$ or G', we indeed have $H, v \vDash \phi$ if exactly one of E_1 and E_2 is safe. If both are safe or both are unsafe, we have $H, v \nvDash \phi$.

The last base case of interest is the case where ϕ is of the form $closed(R)$. This goes again exactly as in the proof for $X = disj$.

We next consider the inductive cases. The cases for the boolean connectives follow readily by induction. Finally, assume ϕ is of the form $\geq_n E.\phi_1$. By induction, there are two possibilities for ϕ_1:

[4] Actually, $[\![E]\!]^G$ always contains $V \times V$ in this case, but we do not need this.

- If $[\![\phi_1]\!]^G \cap V = \emptyset$ then $[\![\phi]\!]^G \cap V = \emptyset$ since path expressions can only reach nodes in some graph from nodes in that graph.
- If $[\![\phi_1]\!]^H \supseteq V$, we distinguish three cases using Lemma 3.3. First, when E is id, then if $n = 1$, $[\![\phi]\!]^H \supseteq V$. Otherwise, if $n \neq 1$, then $[\![\phi]\!]^H = \emptyset$. Next, when E is id-free or $E' \cup id$ with E' an id-free path expression, it suffices to show that $\sharp[\![E']\!]^H(v) \geq n$ for all $v \in V$. By Lemma 3.14, we know that $[\![E]\!]^H(v)$ contains $[\![r]\!]^H(v)$ or $[\![r^-]\!]^H(v)$. These sets contain at least $M - 1 \geq m \geq n$ elements as desired. (The number $M - 1$ is reached only when H is G and $v = b$ or $v = a$; otherwise the sets contain M elements.)

The equality $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$ is shown in the same way as in the proof for $X = disj$ (Section 3.2). □

3.4. **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 3.16. *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 $[\![p]\!]^{G_1} = [\![p]\!]^{G_2}$ for every property name p in Σ. Then $[\![E]\!]^{G_1} = [\![E]\!]^{G_2}$ and $[\![\phi]\!]^{G_1} = [\![\phi]\!]^{G_2}$.* □

Theorem 3.1 now follows readily for $X = closed$. 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 3.16 we have $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$, showing that $\mathcal{S}$ does not define Q_{closed}.

Remark 3.17. Lemma 3.16 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, $[\![closed(\emptyset)]\!]^G$, 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.

The reader may wonder if this statement still holds under active domain semantics. In such semantics, which we denote by $[\![\phi]\!]^G_{adom}$, we would view G as an interpretation with domain *not* the whole of N; rather we would take as domain the set $N_G \cup C$, with C the set of constants mentioned in ϕ. When assuming active domain semantics, a modified lemma is required. To see this, consider the graph $G = \{(a, p, b)\}$ and $G' = \{(a, p, b), (a, q, c)\}$. Let ϕ simply be $\top$. We have $[\![\phi]\!]^G_{adom} = \{a, b\}$ and $[\![\phi]\!]^{G'}_{adom} = \{a, b, c\}$, so Lemma 3.16 no longer holds. We can, however, give the following more refined variant of Lemma 3.16:

Lemma 3.18. *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 I_1 and I_2 be interpretations such that $[\![p]\!]^{I_1} = [\![p]\!]^{I_2}$ for every property name p in Σ. Then $[\![E]\!]^{I_1} \cap \Delta^{I_2} \times \Delta^{I_2} = [\![E]\!]^{I_2} \cap \Delta^{I_1} \times \Delta^{I_1}$ and $[\![\phi]\!]^{I_1} \cap \Delta^{I_2} = [\![\phi]\!]^{I_2} \cap \Delta^{I_1}$.* □

The same reasoning as given after Lemma 3.16, now using the new Lemma, then shows that *closed* is still primitive under active domain semantics.

4. Are target-based shape schemas enough?

Lemma 3.16 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 4.1. *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).*

In order to prove this theorem, we first establish the following lemma.

Lemma 4.2. *Let ϕ be a shape and let C be the set of constants mentioned in ϕ. Assume there exists a graph G and a node name $x \notin N_G \cup C$ such that $G, x \vDash \phi$. Then for any graph H and any node name $y \notin N_H \cup C$, also $H, y \vDash \phi$.*

Proof. By induction on ϕ. The case where ϕ is of the form $\{c\}$ cannot occur, and the case where ϕ is $\top$ is trivial.

If ϕ is $\phi_1 \vee \phi_2$ or $\neg\phi_1$, the claim follows readily by induction.

Now assume ϕ is of the form $\geq_n E.\phi_1$. Then there exists $x_1, \dots, x_n$ such that $(x, x_i) \in [\![E]\!]^G$ and $G, x_i \vDash \phi_1$ for $i = 1, \dots, n$. However, since $x \notin N_G$, by Lemma 3.5, all x_i must equal x. Hence, $n = 1$ and $(x, x) \in [\![E]\!]^G$ and $G, x \vDash \phi_1$. By the same Lemma, $(y, y) \in [\![E]\!]^H$, since $y \notin N_H$. Furthermore, by induction, $H, y \vDash \phi_1$. We conclude that $H, y \vDash \phi$ as desired.

Next, assume ϕ is $eq(E, p)$. Since $G, x \vDash \phi$, but $[\![p]\!]^G = \emptyset$ since $x \notin N_G$, also $[\![E]\!]^G(x) = \emptyset$. Then by Lemma 3.5, also $[\![E]\!]^H(y) = \emptyset$, since $y \notin N_H$. Furthermore, also $[\![p]\!]^H(y) = \emptyset$. We conclude that $H, y \vDash \phi$ as desired.

Next assume ϕ is $disj(E, p)$. Then $H, y \vDash \phi$ is clear. Indeed, since $y \notin N_H$, we have $[\![p]\!]^H(y) = \emptyset$.

Finally, assume ϕ is $closed(R)$. Then again $H, y \vDash \phi$ is clear because $y \notin N_H$. □

We can now show the theorem.

Proof of Theorem 4.1. Let ϕ be the validation shape for shape schema $\mathcal{S}$, so that $G \vDash \mathcal{S}$ if and only if $[\![\phi]\!]^G$ 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 $G, v \vDash \phi$, we have $v \in N_G \cup C$. If ϕ is not internal, then, using Lemma 4.2, for every graph G and every node $v \notin N_G \cup C$, we have $G, v \vDash \phi$. 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:

$$\begin{aligned} G \models \mathcal{S} &\Leftrightarrow [\![\phi]\!]^G = \emptyset \\ &\Leftrightarrow [\![\phi]\!]^{G'} = \emptyset && \text{by Lemma 3.16} \\ &\Leftrightarrow G' \models \mathcal{T} && \text{since } \phi \text{ is internal} \\ &\Leftrightarrow [\![\psi]\!]^{G'} = \emptyset \\ &\Leftrightarrow [\![\psi]\!]^{G} = \emptyset && \text{by Lemma 3.16} \\ &\Leftrightarrow G \models \mathcal{T} \end{aligned}$$

□

Remark 4.3. 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

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

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

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

Remark 4.4. Theorem 4.1 fails in the presence of closure constraints. For example, the inclusion $\neg \mathit{closed}(\emptyset) \subseteq \exists r.\top$ 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.

5. Extensions for full equality and disjointness tests

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. The next question we can ask is whether allowing "full" equality or disjointness tests, i.e., allowing a general path expression for E_2, strictly increases the expressive power. Within the community there are indeed plans to extend SHACL in this direction [Knu21, Jak22].

When we allow for such "full" equality and disjointness tests, it gives rise to two new features: *full-eq* and *full-disj*. Formally, we extend the grammar of shapes with two new constructs: $eq(E_1, E_2)$ and $disj(E_1, E_2)$.

Remark 5.1. We extend Remark 2.3 by noting that in real SHACL, we cannot explicitly write the shapes $eq(id, id)$ and $disj(id, id)$. However, these shapes are equivalent to $\top$ and $\neg\top$ respectively.

We are going to show that each of these new features strictly adds expressive power. Concretely, we introduce the following classes of graphs.

Full equality: $Q_{\mathit{full\text{-}eq}}$ is the class of graphs where all objects of a property name p do not have the same subjects for p and q. Note that $Q_{\mathit{full\text{-}eq}}$ is definable in $\mathcal{L}(\mathit{full\text{-}eq})$ by the single, target-based, inclusion statement $\exists p^-.\top \subseteq \neg eq(p^-, q^-)$.

Full disjointness: $Q_{\textit{full-disj}}$ is the class of graphs where all objects of a property name p do not have disjoint sets of subjects for p and q. Note that $Q_{\textit{full-disj}}$ is definable in $\mathcal{L}(\textit{full-disj})$ by the single, target-based, inclusion statement $\exists p^-.\top \subseteq \neg \textit{disj}(p^-, q^-)$.

In the spirit of Theorem 3.1, we are now going to show the following:

Theorem 5.2. *$Q_{\textit{full-eq}}$ is not definable in $\mathcal{L}(\textit{eq}, \textit{full-disj}, \textit{closed})$ and $Q_{\textit{full-disj}}$ is not definable in $\mathcal{L}(\textit{disj}, \textit{full-eq}, \textit{closed})$.*

These two non-definability results are proven in the following Sections 5.1 and 5.2. Then in Section 5.3 we will reconsider the non-definability results for non-full equality and disjointness from Theorem 3.1 in the new light of their full versions.

5.1. **Full equality.** We present here the proof for the primitivity of full equality tests. The general strategy is the same as in Section 3, where again we will prove appropriate versions of Proposition 3.2.

We begin by defining the graphs G and G' formally. Note that, as desired, G' belongs to $Q_{\textit{full-eq}}$ but G does not.

Definition 5.3. $G_{\textit{full-eq}}(\Sigma, m)$ Let Σ be a finite vocabulary and let $m \geq 3$ be a natural number. Let $A = \{a_1, \dots, a_m\}$, $B = \{b_1, \dots, b_m\}$ and $C = \{c_1, \dots, c_m\}$ be three disjoint sets of nodes, disjoint from Σ. We define the graph $G_{\textit{full-eq}}(\Sigma, m)$ to be $[\![p]\!]^G = C \times (A \cup B)$ and $[\![q]\!]^G = C \times A \cup \{(c_i, b_j) \mid i \neq j \in \{1, \dots, m\}\}$.

Definition 5.4. $G'_{\textit{full-eq}}(\Sigma, m)$ We define the graph $G'_{\textit{full-eq}}(\Sigma, m)$ like $G_{\textit{full-eq}}(\Sigma, m)$ but $[\![q]\!]^G = \{(c_i, a_j) \mid i \neq j \in \{1, \dots, m\}\} \cup \{(c_i, b_j) \mid i \neq j \in \{1, \dots, m\}\}$.

We identify the possible types of strings on the graphs $G_{\textit{full-eq}}(\Sigma, m)$ and $G'_{\textit{full-eq}}(\Sigma, m)$ as follows.

Lemma 5.5. *Let Σ be a vocabulary. Let $m \geq 3$ be a natural number. Let G be $G_{\textit{full-eq}}(\Sigma, m)$ and let G' be $G'_{\textit{full-eq}}(\Sigma, m)$. The only possibilities for a string s evaluated on G and G' are the following:*

(1) $[\![s]\!]^G = [\![p]\!]^G = [\![s]\!]^{G'} = C \times (A \cup B)$.
(2) $[\![s]\!]^G = [\![q]\!]^G = (C \times A) \cup \{(c_i, b_j) \mid i \neq j \in \{1, \dots, m\})\}$ *and*
$[\![s]\!]^{G'} = [\![q]\!]^{G'} = \{(c_i, a_j) \mid i \neq j \in \{1, \dots, m\}\} \cup \{(c_i, b_j) \mid i \neq j \in \{1, \dots, m\}\}$.
(3) $[\![s]\!]^G = [\![p^-]\!]^G = [\![s]\!]^{G'} = (A \cup B) \times C$.
(4) $[\![s]\!]^G = [\![q^-]\!]^G = (A \times C) \cup \{(b_i, c_j) \mid i \neq j \in \{1, \dots, m\})\}$ *and*
$[\![s]\!]^{G'} = [\![q^-]\!]^{G'} = \{(a_i, c_j) \mid i \neq j \in \{1, \dots, m\})\} \cup \{(b_i, c_j) \mid i \neq j \in \{1, \dots, m\})\}$.
(5) $[\![s]\!]^G = [\![s]\!]^{G'} = C \times C$.
(6) $[\![s]\!]^G = [\![s]\!]^{G'} = (A \cup B) \times (A \cup B)$.
(7) $[\![s]\!]^G = [\![s]\!]^{G'} = id$.
(8) $[\![s]\!]^G = [\![s]\!]^{G'} = \emptyset$.

Proof. We show this by systematically enumerating all strings until no new binary relations can be found. Note that we only enumerate over strings that alternate between property names and reversed property names. Indeed, all other strings evaluate to the empty relation on both G and G'. Every time we encounter new binary relations, we put the string in boldface.

s	$[\![s]\!]^G$	$[\![s]\!]^{G'}$
$\mathbf{id}$	id	id
$\mathbf{p}$	$C \times (A \cup B)$	$C \times (A \cup B)$
$\mathbf{q}$	$(C \times A) \cup$ $\{(c_i, b_j) \mid i \neq j \in \{1, \ldots, m\})\}$	$\{(c_i, a_j) \mid i \neq j \in \{1, \ldots, m\}\} \cup$ $\{(c_i, b_j) \mid i \neq j \in \{1, \ldots, m\}\}$
$\mathbf{p}^-$	$(A \cup B) \times C$	$(A \cup B) \times C$
$\mathbf{q}^-$	$(A \times C) \cup$ $\{(b_i, c_j) \mid i \neq j \in \{1, \ldots, m\})\}$	$\{(a_i, c_j) \mid i \neq j \in \{1, \ldots, m\})\} \cup$ $\{(b_i, c_j) \mid i \neq j \in \{1, \ldots, m\})\}$
$\mathbf{p}/\mathbf{p}^-$	$C \times C$	$C \times C$
p/q^-	$C \times C$	$C \times C$
q/p^-	$C \times C$	$C \times C$
q/q^-	$C \times C$	$C \times C$
$\mathbf{p}^-/\mathbf{p}$	$(A \cup B) \times (A \cup B)$	$(A \cup B) \times (A \cup B)$
p^-/q	$(A \cup B) \times (A \cup B)$	$(A \cup B) \times (A \cup B)$
q^-/p	$(A \cup B) \times (A \cup B)$	$(A \cup B) \times (A \cup B)$
q^-/q	$(A \cup B) \times (A \cup B)$	$(A \cup B) \times (A \cup B)$
$p/p^-/p$	$C \times (A \cup B)$	$C \times (A \cup B)$
$p/p^-/q$	$C \times (A \cup B)$	$C \times (A \cup B)$
$p^-/p/p^-$	$(A \cup B) \times C$	$(A \cup B) \times C$
$p^-/p/q^-$	$(A \cup B) \times C$	$(A \cup B) \times C$

□

We are now ready to prove the key proposition.

Proposition 5.6. *Let Σ be a vocabulary. Let $m \geq 3$ be a natural number. Let $V = A \cup B \cup C$ be the common set of nodes of the graphs $G = G_{\mathit{full\text{-}eq}}(\Sigma, m)$ and $G' = G'_{\mathit{full\text{-}eq}}(\Sigma, m)$. For all shapes ϕ over Σ counting to at most $m-1$, we have $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$. Moreover,*

- $[\![\phi]\!]^G \cap V = A \cup B$, *or*
- $[\![\phi]\!]^G \cap V = C$, *or*
- $[\![\phi]\!]^G \cap V = V$, *or*
- $[\![\phi]\!]^G \cap V = \emptyset$.

Proof. By induction on the structure of ϕ. For the base cases, if ϕ is $\top$ then $[\![\top]\!]^G = [\![\top]\!]^{G'} = N$ and $N \cap V = V$. If ϕ is $\{c\}$, then $[\![\{c\}]\!]^G = [\![\{c\}]\!]^{G'} = \{c\}$ and $\{c\} \cap V = \emptyset$ since $c \in \Sigma$ and $V \cap \Sigma = \emptyset$.

If ϕ is $\mathit{closed}(Q)$, we consider the possibilities for Q. If Q does not contain both p and q, then clearly $[\![\phi]\!]^G \cap V = [\![\phi]\!]^{G'} \cap V = A \cup B$. Otherwise, $[\![\phi]\!]^G = [\![\phi]\!]^{G'} = N$.

Before considering the remaining cases, we observe the following symmetries:

- All elements of A are symmetrical in G. This is obvious from the definition of G.
- Also in G', all elements of A are symmetrical. Indeed, for any $a_i \neq a_j$ in A, the function that swaps a_i and a_j, as well as c_i and c_j, is an automorphism of G'.
- Similarly, all elements of B are symmetrical in G, and also in G'.
- Moreover, we see that all elements of C are symmetrical in G, and in G'.
- Finally, in G', any a_i and b_j are symmetrical. Indeed, the function that swaps a_i and b_i is clearly an automorphism of G'. In turn, b_i and b_j are symmetrical by the above.

Therefore, we are only left to show:

(i) For any $a \in A$ and $b \in B$, we have $G, a \models \phi \iff G, b \models \phi$,

(ii) For any $a \in A$, we have $G, a \vDash \phi \iff G', a \vDash \phi$, and
(iii) For any $c \in C$, we have $G, c \vDash \phi \iff G', c \vDash \phi$.
(iv) For any $x \notin V$, we have $G, x \vDash \phi \iff G', x \vDash \phi$.

Note that then also for any $b \in B$, we have $G, b \vDash \phi \iff G', b \vDash \phi$ because for any $a \in A$ and $b \in B$, we have $G, b \vDash \phi \overset{\text{(i)}}{\iff} G, a \vDash \phi \overset{\text{(ii)}}{\iff} G', a \vDash \phi \overset{\textit{symmetry}}{\iff} G', b \vDash \phi$.

Consider the case where ϕ is $eq(E, r)$. We verify (i), (ii), (iii), and (iv).

(i) By definition of G, $[\![r]\!]^G(a) = [\![r]\!]^G(b) = \emptyset$ for any property name r. Therefore we need to show $[\![E]\!]^G(a) = \emptyset \iff [\![E]\!]^G(b) = \emptyset$. By Lemma 3.7 we know there is a set of strings U equivalent to E in both G and G'. By Lemma 5.5 there are only 8 types of strings. We observe from Lemma 5.5 that for every U, $\bigcup_{s \in U} [\![s]\!]^G(a)$ is empty whenever U only contains strings of type 1, 2, 5, or 8. These are also exactly the U s.t. $\bigcup_{s \in U} [\![s]\!]^G(b)$ is empty.

(ii) Furthermore, these are also exactly the sets of strings U s.t. $\bigcup_{s \in U} [\![s]\!]^{G'}(a)$ is empty. Therefore, as $[\![r]\!]^{G'}(a) = \emptyset$, we have $G', a \vDash \phi$.

(iii) Assume $G, c \vDash \phi$. We consider the possibilities for r. First, suppose $r = p$. The sets of strings U s.t. $\bigcup_{s \in U} [\![s]\!]^G(c) = [\![p]\!]^G(c)$ contain strings of type 1 but not strings of type 5 or 7. These are also exactly the U s.t. $\bigcup_{s \in U} [\![s]\!]^{G'}(c) = [\![p]\!]^{G'}(c)$.

Next, suppose $r = q$. The sets of strings U s.t. $\bigcup_{s \in U} [\![s]\!]^G(c) = [\![p]\!]^G(c)$ contain strings of type 2 but not strings of type 1, 5 or 7. These are also exactly the types of strings s.t. $\bigcup_{s \in U} [\![s]\!]^{G'}(c) = [\![q]\!]^{G'}(c)$.

Finally, if r is any other property name, then $[\![r]\!]^G(c) = [\![r]\!]^{G'}(c) = \emptyset$. This is the case when U does not contain any strings of type 1, 2, 3, or 7. These are also exactly the types of strings U s.t. $\bigcup_{s \in U} [\![s]\!]^{G'}(c) = \emptyset$.

(iv) Let $x \in N - V$. Clearly $[\![r]\!]^G(x) = [\![r]\!]^{G'}(x) = \emptyset$. By Lemma 3.5, if E is safe, then $[\![E]\!]^G(x) = [\![E]\!]^{G'}(x) = \emptyset$. Therefore $G, x \vDash \phi$ and $G', x \vDash \phi$. On the other hand, whenever E is unsafe, $[\![E]\!]^G(x) = [\![E]\!]^{G'}(x) = \{x\} \neq \emptyset$. Therefore, $G, x \nvDash \phi$ and $G', x \nvDash \phi$.

Next, consider the case where ϕ is $disj(E_1, E_2)$. We again verify (i), (ii), (iii), and (iv).

(i) Assume $G, a \vDash disj(E_1, E_2)$. This can only be the case when the corresponding sets of strings U_1 and U_2 are of the following form. U_1 can consist only of strings of type 3, 4, 1, 2, 5, and 8 (Here, types 1, 2, 5 and 8 evaluate to empty from a as already seen above). U_2 can then only consist of strings of type 6, 7, 1, 2, 5, and 8 (or vice versa). These are also the only cases where $G, b \vDash disj(E_1, E_2)$.

(ii) Exactly the same situation occurs in G' and these are then also the only cases where $G', a \vDash disj(E_1, E_2)$.

(iii) Assume $G, c \vDash disj(E_1, E_2)$. This can only be the case when the corresponding sets of strings U_1 and U_2 are of the following form. U_1 can consist only of strings of type 1, 2, 3, 4, 6, and 8 (Here, types 3, 4, 6, and 8 evaluate to empty from c as already seen above). U_2 can then only consist of strings of type 5, 7, 3, 4, 6, and 8. We observe that this is also the case in G'.

(iv) Let $x \in N - V$. Whenever E_1 is safe, by Lemma 3.5 $[\![E_1]\!]^G(x) = [\![E_1]\!]^{G'}(x) = \emptyset$. Therefore, $G, x \vDash \phi$ and $G', x \vDash \phi$. Clearly, the same holds whenever E_2 is safe. When both E_1 and E_2 are unsafe, $[\![E_1]\!]^G(x) = [\![E_1]\!]^{G'}(x) = \{x\} \neq \emptyset$. Therefore, $G, x \nvDash \phi$ and $G', x \nvDash \phi$.

The cases where ϕ is $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ or $\neg\phi_1$ are handled by induction in a straightforward manner.

Lastly, we consider the case where ϕ is $\geq_n E.\phi_1$.

(i) Assume $G, a \vDash \phi$. Then, there exist distinct $x_1, \ldots, x_n$ s.t. $(a, x_i) \in [\![E]\!]^G$ and $G, x_i \vDash \phi_1$ for $1 \leq i \leq n$. Again, by Lemma 3.7 we know there is a set of strings U equivalent to E in both G and G'. By Lemma 5.5 there are only 8 types of strings. By induction, we consider three cases.

First, if $[\![\phi_1]\!]^G \cap V = A \cup B$, then $x_i, \ldots, x_n \in A \cup B$. Therefore, we know U must at least contain strings of type 6 or 7. Suppose U contains strings of type 6. Then, we verify that $\sharp(\bigcup_{s \in U} [\![s]\!]^G(b) \cap (A \cup B)) \geq m - 1 \geq n$. Whenever U contains strings of type 7, and not of type 6, we know $n = 1$ and clearly $\sharp(\bigcup_{s \in U} [\![s]\!]^G(b) \cap (A \cup B)) = 1$.

Next, if $[\![\phi_1]\!]^G \cap V = C$, then $x_i, \ldots, x_n \in C$. Therefore, we know U must at least contain strings of type 3 or 4. Whenever U contains 3 or 4, we verify that $\sharp(\bigcup_{s \in U} [\![s]\!]^G(b) \cap C) \geq m - 1 \geq n$.

Next, if $[\![\phi_1]\!]^G \cap V = V$, then $x_i, \ldots, x_n \in V$. Therefore, we know U must at least contain strings of type 3, 4, 6 or 7. All these types have already been handled in the previous two cases.

Finally, the case where $[\![\phi_1]\!]^G \cap V = \emptyset$ cannot occur as $n > 0$.

The sets of strings U above are also exactly the sets used to argue the implication from right to left.

(ii) For every case of U above, for every inductive case of ϕ_1, we can also verify that $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(a) \cap (A \cup B)) \geq m - 1 \geq n$, $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(a) \cap C) \geq m - 1 \geq n$, and $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(a) \cap V) \geq m - 1 \geq n$.

(iii) Assume $G, c \vDash \phi$. Then, there exist distinct $x_1, \ldots, x_n$ s.t. $(c, x_i) \in [\![E]\!]^G$ and $G, x_i \vDash \phi_1$ for $1 \leq i \leq n$. By induction, we consider three cases.

First, if $[\![\phi_1]\!]^G \cap V = A \cup B$, then $x_i, \ldots, x_n \in A \cup B$. Therefore, we know U must at least contain strings of type 1 or 2. Whenever U contains 1 or 2, we verify that $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(c) \cap (A \cup B)) \geq m \geq n$.

Next, if $[\![\phi_1]\!]^G \cap V = C$, then $x_i, \ldots, x_n \in C$. Therefore, we know U must at least contain strings of type 5 or 7. Whenever U contains 5, we verify that $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(c) \cap C) \geq m \geq n$. Otherwise, whenever U contains strings of type 7, and not of type 5, we know $n = 1$ and clearly $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(c) \cap C) = 1$.

Next, if $[\![\phi_1]\!]^G \cap V = V$, then $x_i, \ldots, x_n \in V$. Therefore, we know U must at least contain strings of type 1, 2, 5 or 7. All these types have already been handled in the previous two cases.

Finally, the case where $[\![\phi_1]\!]^G \cap V = \emptyset$ cannot occur as $n > 0$.

The sets of strings U above are also exactly the sets used to argue the implication from right to left.

(iv) For the direction from left to right, take $x \in [\![\phi]\!]^G \setminus V$. Since $G, x \vDash \phi$, there exists $y_1, \ldots, y_n$ s.t. $(x, y_i) \in [\![E]\!]^G$ and $G, y_i \vDash \phi_1$ for $i = 1, \ldots, n$. However, since $x \notin V$, by Lemma 3.5, all y_i must equal x. Hence, $n = 1$ and $(x, x) \in [\![E]\!]^G$ and $G, x \vDash \phi_1$. Then again, by the same Lemma, $(x, x) \in [\![E]\!]^{G'}$, since G and G' have the same set of nodes V. Moreover, by induction, $G', x \vDash \phi_1$. We conclude $G', x \vDash \phi$ as desired. The direction from right to left is argued symmetrically. □

5.2. **Full disjointness.** We present here the proof for the primitivity of full disjointness tests. The general strategy is the same as in Section 5.1.

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

Definition 5.7. $G_{\mathit{full\text{-}disj}}(\Sigma, m)$ Let m be a natural number that is a multiple of 8. Let $A = \{a_1, \dots, a_m\}$, $B = \{b_1, \dots, b_m\}$ and $C = \{c_1, \dots, c_m\}$ be three disjoint sets of nodes, disjoint from Σ. For any $i \leq j$, we write $a_{i\to j}$ to denote the set

$$\{a_{1+(i-1+l \mod m)} \mid 0 \leq l \leq j-i\}$$

We define $b_{i\to j}$ and $c_{i\to j}$ analogously.

We define the graph $G_{\mathit{full\text{-}disj}}(\Sigma, m)$ by:

$$[\![p]\!]^G(c_i) = a_{i\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1} \text{ and}$$
$$[\![q]\!]^G(c_i) = a_{i-\frac{m}{2}\to i-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1} \text{ for } 1 \leq i \leq m$$

The p and q relations are visualized in Figure 3.

To give an example for our notation, suppose $m = 8$. Then,

$$a_{2\to 5} = \{a_{1+(1+l \mod 8)} \mid 0 \leq l \leq 3\} = \{a_2, a_3, a_4, a_5\}$$
$$a_{7\to 10} = \{a_{1+(6+l \mod 8)} \mid 0 \leq l \leq 3\} = \{a_7, a_8, a_1, a_2\}, \text{ and}$$
$$a_{-4\to -1} = \{a_{1+(-5+l \mod 8)} \mid 0 \leq l \leq 3\} = \{a_4, a_5, a_6, a_7\}$$

Definition 5.8. $G'_{\mathit{full\text{-}disj}}(\Sigma, m)$ We define the graph $G'_{\mathit{full\text{-}disj}}(\Sigma, m)$ on the same nodes as $G_{\mathit{full\text{-}disj}}(\Sigma, m)$, with the only difference being the relationship of the p- and q-edges from C to A:

$$[\![p]\!]^G(c_i) = a_{i-\frac{m}{8}\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1} \text{ and}$$
$$[\![q]\!]^G(c_i) = a_{i-\frac{m}{2}\to i+\frac{m}{8}-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1} \text{ for } 1 \leq i \leq m$$

The p and q relations are visualized in Figure 3.

Important to the intuition behind these graphs is the overlap generated by the inverse p- and q-edges. As demonstrated in Figure 4, in graph $G = G_{\mathit{full\text{-}disj}}(\Sigma, m)$, the set of c nodes reached from a nodes with inverse p edges is disjoint from the set of c nodes reached with inverse q edges. This is not the case for b nodes: there, these sets overlap by precisely one fourth of the c nodes. For graph $G' = G'_{\mathit{full\text{-}disj}}(\Sigma, m)$, the sets of c nodes reachable by inverse p and q edges overlap for both a and b nodes.

We precisely characterize the behavior of strings on the graphs G and G' as follows.

Lemma 5.9. *Let Σ be a vocabulary. Let m be a natural number that is a multiple of 8. Let G be $G_{\mathit{full\text{-}disj}}(\Sigma, m)$ and let G' be $G'_{\mathit{full\text{-}disj}}(\Sigma, m)$. The only possibilities for a string s evaluated on G and G' are the following:*

(1) $[\![s]\!]^G = [\![p]\!]^G = \bigcup_{i\in\{1,\dots,m\}}\{c_i\} \times (a_{i\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1})$ *and*
$[\![s]\!]^{G'} = [\![p]\!]^{G'} = \bigcup_{i\in\{1,\dots,m\}}\{c_i\} \times (a_{i-\frac{m}{8}\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1})$;
(2) $[\![s]\!]^G = [\![q]\!]^G = \bigcup_{i\in\{1,\dots,m\}}\{c_i\} \times (a_{i-\frac{m}{2}\to i-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1})$ *and*
$[\![s]\!]^{G'} = [\![q]\!]^G = \bigcup_{i\in\{1,\dots,m\}}\{c_i\} \times (a_{i-\frac{m}{2}\to i+\frac{m}{8}-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1})$;
(3) $[\![s]\!]^G = [\![p^-]\!]^G = \bigcup_{i\in\{1,\dots,m\}}(\{a_i\} \times c_{i-\frac{m}{2}+1\to i}) \cup (\{b_i\} \times c_{i-\frac{m}{2}+1\to i+\frac{m}{8}})$ *and*
$[\![s]\!]^{G'} = [\![p^-]\!]^{G'} = \bigcup_{i\in\{1,\dots,m\}}(\{a_i\} \times c_{i-\frac{m}{2}+1\to i+\frac{m}{8}}) \cup (\{b_i\} \times c_{i-\frac{m}{2}+1\to i+\frac{m}{8}})$;

(4) $[\![s]\!]^G = [\![q^-]\!]^G = \bigcup_{i\in\{1,\dots,m\}}(\{a_i\}\times c_{i+1\to i+\frac{m}{2}}) \cup (\{b_i\}\times c_{i-\frac{m}{8}+1\to i+\frac{m}{2}})$ *and*
$[\![s]\!]^{G'} = [\![q^-]\!]^{G'} = \bigcup_{i\in\{1,\dots,m\}}(\{a_i\}\times c_{i-\frac{m}{8}+1\to i+\frac{m}{2}}) \cup (\{b_i\}\times c_{i-\frac{m}{8}+1\to i+\frac{m}{2}})$;

(5) $[\![s]\!]^G = [\![s]\!]^{G'} = C \times C$;

(6) $[\![s]\!]^G = [\![s]\!]^{G'} = (A\cup B) \times (A \cup B)$;

(7) $[\![s]\!]^G = [\![s]\!]^{G'} = C \times (A \cup B)$;

(8) $[\![s]\!]^G = [\![s]\!]^{G'} = (A\cup B) \times C$;

(9) $[\![s]\!]^G = [\![s]\!]^{G'} = id$; *or*

(10) $[\![s]\!]^G = [\![s]\!]^{G'} = \emptyset$

The first four types of strings are visualized in Figure 3 and Figure 4.

Proof. The proof is performed as in the proof of Lemma 5.5. We now have the following table:

s	$[\![s]\!]^G$	$[\![s]\!]^{G'}$
$\mathbf{id}$	id	id
$\mathbf{p}$	type 1	type 1
$\mathbf{q}$	type 2	type 2
$\mathbf{p}^-$	type 3	type 3
$\mathbf{q}^-$	type 4	type 4
$\mathbf{p}/\mathbf{p}^-$	$C\times C$	$C\times C$
p/q^-	$C\times C$	$C\times C$
q/p^-	$C\times C$	$C\times C$
q/q^-	$C\times C$	$C\times C$
$\mathbf{p}^-/\mathbf{p}$	$(A\cup B)\times(A\cup B)$	$(A\cup B)\times(A\cup B)$
p^-/q	$(A\cup B)\times(A\cup B)$	$(A\cup B)\times(A\cup B)$
q^-/p	$(A\cup B)\times(A\cup B)$	$(A\cup B)\times(A\cup B)$
q^-/q	$(A\cup B)\times(A\cup B)$	$(A\cup B)\times(A\cup B)$
$\mathbf{p}/\mathbf{p}^-/\mathbf{p}$	$C\times(A\cup B)$	$C\times(A\cup B)$
$p/p^-/q$	$C\times(A\cup B)$	$C\times(A\cup B)$
$\mathbf{p}^-/\mathbf{p}/\mathbf{p}^-$	$(A\cup B)\times C$	$(A\cup B)\times C$
$p^-/p/q^-$	$(A\cup B)\times C$	$(A\cup B)\times C$
$p/p^-/p/p^-$	$C\times C$	$C\times C$
$p/p^-/p/q^-$	$C\times C$	$C\times C$
$p^-/p/p^-/p$	$(A\cup B)\times(A\cup B)$	$(A\cup B)\times(A\cup B)$
$p^-/p/p^-/q$	$(A\cup B)\times(A\cup B)$	$(A\cup B)\times(A\cup B)$

□

We are ready to present our key Proposition.

Proposition 5.10. *Let Σ be a vocabulary. Let m be a natural number and a multiple of 8. Let $V = A \cup B \cup C$ be the common set of nodes of the graphs $G = G_{\textit{full-disj}}(\Sigma, m)$ and $G' = G'_{\textit{full-disj}}(\Sigma, m)$. For all shapes ϕ over Σ counting to at most $\frac{m}{2}$, we have $[\![\phi]\!]^G = [\![\phi]\!]^{G'}$. Moreover,*

- $[\![\phi]\!]^G \cap V = A \cup B$, *or*
- $[\![\phi]\!]^G \cap V = C$, *or*
- $[\![\phi]\!]^G \cap V = V$, *or*
- $[\![\phi]\!]^G \cap V = \emptyset$.

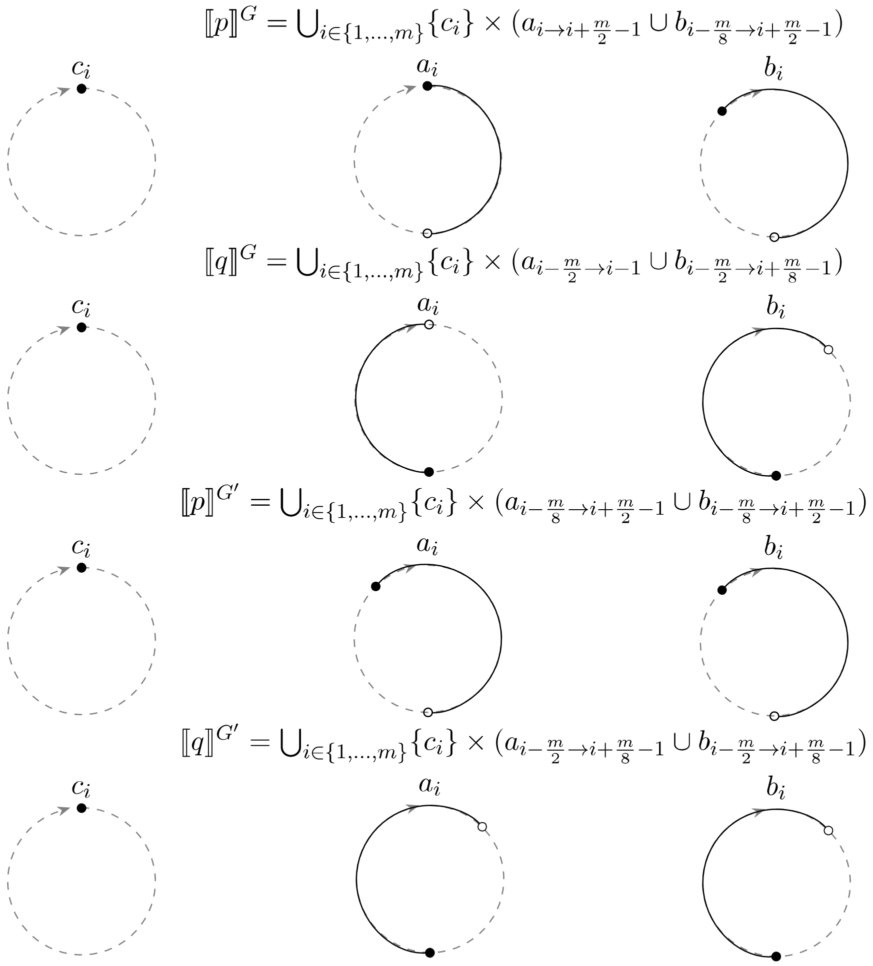

Figure 3: Illustration of the p and q relations in graphs $G = G_{\textit{full-disj}}(\Sigma, m)$ and $G' = G'_{\textit{full-disj}}(\Sigma, m)$

Proof. By induction on the structure of ϕ. For the base cases, if ϕ is $\top$ then $[\![\top]\!]^G = [\![\top]\!]^{G'} = N$ and $N \cap V = V$. If ϕ is $\{c\}$, then $[\![\{c\}]\!]^G = [\![\{c\}]\!]^{G'} = \{c\}$ and $\{c\} \cap V = \emptyset$ since $c \in \Sigma$ and $V \cap \Sigma = \emptyset$.

If ϕ is $\textit{closed}(Q)$, we consider the possibilities for Q. If Q does not contain both p and q, then clearly $[\![\phi]\!]^G \cap V = [\![\phi]\!]^{G'} \cap V = A \cup B$. Otherwise, $[\![\phi]\!]^G = [\![\phi]\!]^{G'} = N$.

Before considering the remaining cases, we observe the following symmetries:

- In both G and G', all elements of A are symmetrical, as are all elements of B, and all elements of C. Indeed, for any $i \in \{1, \dots, m\}$, the function that maps x_i to $x_{1+i \bmod m}$ where x_i is a_i, b_i or c_i, is clearly an automorphism of G and also of G'.
- Furthermore, in G', any a_i and b_j are symmetrical. Indeed, the function that swaps every a_i with b_i is an automorphism of G'. (We already know that b_i and b_j are symmetrical by the above.)

Therefore, we are only left to show:

(i) For any $a \in A$ and $b \in B$, we have $G, a \models \phi \iff G, b \models \phi$,

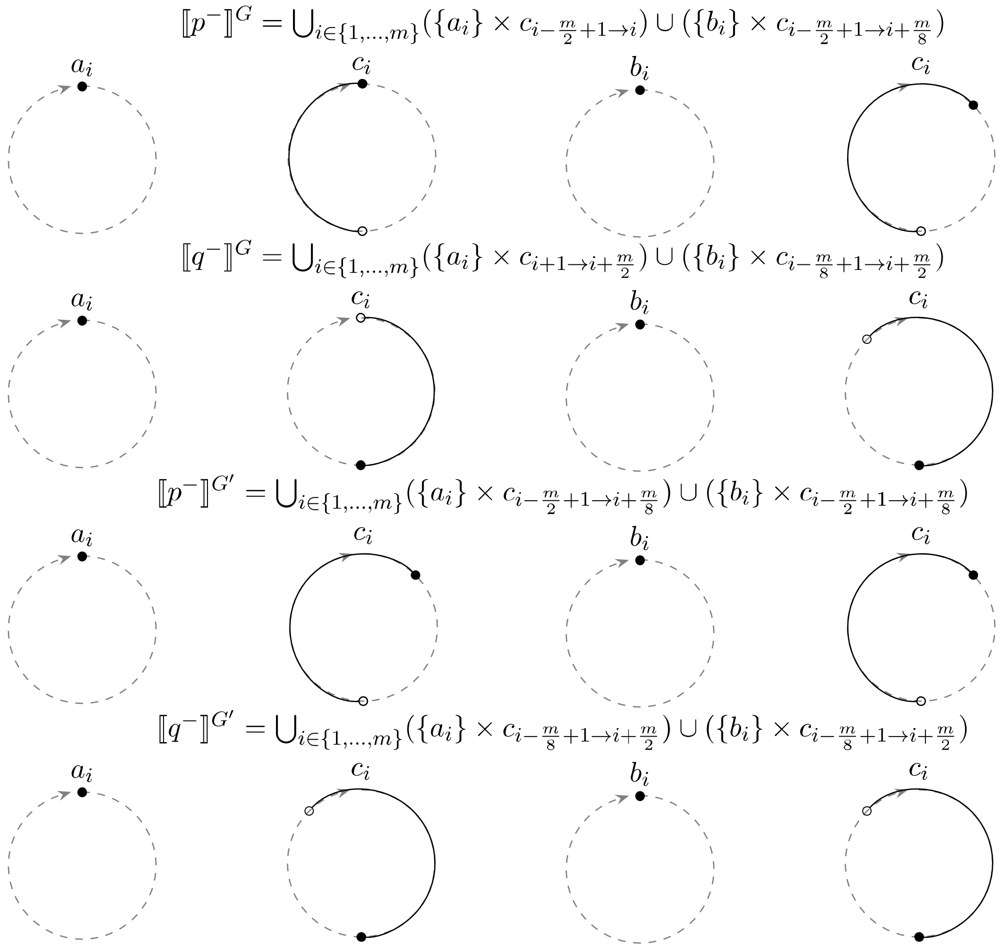

Figure 4: Illustration of the p^- and q^- relations in graphs $G = G_{\mathit{full\text{-}disj}}(\Sigma, m)$ and $G' = G'_{\mathit{full\text{-}disj}}(\Sigma, m)$

(ii) For any $a \in A$, we have $G, a \vDash \phi \iff G', a \vDash \phi$, and
(iii) For any $c \in C$, we have $G, c \vDash \phi \iff G', c \vDash \phi$.
(iv) For any $x \notin V$, we have $G, x \vDash \phi \iff G', x \vDash \phi$.

Note that then also for any $b \in B$, we have $G, b \vDash \phi \iff G', b \vDash \phi$ because for any $a \in A$ and $b \in B$, we have $G, b \vDash \phi \overset{\text{(i)}}{\iff} G, a \vDash \phi \overset{\text{(ii)}}{\iff} G', a \vDash \phi \overset{\mathit{symmetry}}{\iff} G', b \vDash \phi$.

Consider the case where ϕ is $\mathit{disj}(E, r)$. We verify (i), (ii), (iii), and (iv). First, to see that (i) and (ii) hold, we observe that $[\![r]\!]^G(a) = [\![r]\!]^G(b) = [\![r]\!]^{G'}(a) = [\![r]\!]^{G'}(b) = \emptyset$. Therefore, $G, a \vDash \phi$, $G, b \vDash \phi$, $G', a \vDash \phi$, and $G', b \vDash \phi$ always hold, showing (i) and (ii).

Next, to show (iii) where $r = p$, assume $G, c \vDash \mathit{disj}(E, p)$. By Lemma 3.7 we know there is a set of strings U equivalent to E in both G and G'. By Lemma 5.9 there are only 10 types of strings. We observe from Lemma 5.9 that for every U, $\bigcup_{s\in U}[\![s]\!]^G(c)$ is disjoint from $[\![p]\!]^G(c)$ whenever U does not contain strings of type 1, 2, or 7. These are also exactly the U s.t. $\bigcup_{s\in U}[\![s]\!]^{G'}(c)$ is disjoint from $[\![p]\!]^{G'}(c)$.

Next, to show (iii) where $r = q$, assume $G, c \vDash disj(E, q)$. We observe from Lemma 5.9 that for every U, $\bigcup_{s\in U}[\![s]\!]^G(c)$ is disjoint from $[\![q]\!]^G(c)$ whenever U does not contain strings of type 1, 2, or 7. These are also exactly the U s.t. $\bigcup_{s\in U}[\![s]\!]^{G'}(c)$ is disjoint from $[\![q]\!]^{G'}(c)$.

For every other property name r, $[\![r]\!]^G(a) = [\![r]\!]^G(b) = \emptyset$. Therefore, $G, c \vDash \phi$ and $G', c \vDash \phi$ always hold.

Finally, we show (iv) by observing that for any $x \in N \setminus V$, $[\![r]\!]^G(x) = [\![r]\!]^{G'}(x) = \emptyset$. Therefore, $G, x \vDash \phi$ and $G', x \vDash \phi$ always hold.

Next, consider the case where ϕ is $eq(E_1, E_2)$. We again verify (i), (ii), (iii), and (iv).

We show (i) by using a canonical labeling argument. For any two sets U_1 and U_2 of types, we call U_1 and U_2 equivalent in $a \in A$ if $\bigcup_{s\in U_1}[\![s]\!]^G(a) = \bigcup_{s\in U_2}[\![s]\!]^G(a)$. Similarly, we define when U_1 and U_2 are equivalent in b or c.

We can canonically label the equivalence classes in a as follows. Let $U = \{u_1, \dots, u_l\}$ and $u_1 < \cdots < u_l$ with each $u_i \in \{1, \dots, 10\}$ a type.

There are only six unique singleton sets namely $\{1\}$, $\{3\}$, $\{4\}$, $\{6\}$, $\{8\}$, and $\{9\}$. Replace each u_i by their singleton representative $\overline{u_i}$. In $\{\overline{u_1}, \dots, \overline{u_l}\}$, reorder and remove duplicates to obtain an equivalent set $\{u'_1, \dots, u'_{l'}\}$.

If $l' = 1$, we are done. Otherwise, we enumerate all nonequivalent 2-element sets that are not equivalent to a singleton: there are again six of those, namely $\{3, 6\}$, $\{3, 9\}$, $\{4, 6\}$, $\{4, 9\}$, $\{6, 8\}$, and $\{8, 9\}$.

Replace u'_1 and u'_2 by either $\{u''\}$ in case $\{u'_1, u'_2\}$ is equivalent to a singleton; otherwise replace u'_1 and u'_2 by their equivalent 2-element set $\{u''_1, u''_2\}$. If $l' = 2$, we are again done.

We can repeat this process. However, it turns out that there are no 3-element sets that are not equivalent to a singleton or a 2-element set. Hence, there are only 12 representatives. The enumeration process is shown in Table 3, giving the representative for equivalence in a as well as for equivalence in b. Crucially, in filling the table, we observe every set U has the same representative for equivalence in a as for equivalence in b.

Next, (ii) is shown in an analogous manner, where the enumeration process is again shown in Table 3.

Next, (iii) is again shown with an analogous manner, where the enumeration process is shown in Table 4.

To show (iv), assume $x \notin V$. If both E_1 and E_2 are safe, then by Lemma 3.5 $[\![E_1]\!]^G(x) = [\![E_2]\!]^{G'}(x) = \emptyset$. Thus, $G, x \vDash \phi$ and $G', x \vDash \phi$. If both E_1 and E_2 are unsafe, then by Lemma 3.5 $[\![E_1]\!]^G(x) = [\![E_2]\!]^{G'}(x) = \{x\}$. Thus, $G, x \vDash \phi$ and $G', x \vDash \phi$. However, whenever only one of E_1 and E_2 is safe, clearly $G, x \nvDash \phi$ and $G', x \nvDash \phi$.

The cases where ϕ is $\phi_1 \wedge \phi_2$, $\phi_1 \vee \phi_2$ or $\neg\phi_1$ are handled by induction in a straightforward manner.

Lastly, we consider the case where ϕ is $\geq_n E.\psi$.

(i) Assume $G, a \vDash \phi$. Then, there exist distinct $x_1, \dots, x_n$ s.t. $(a, x_i) \in [\![E]\!]^G$ and $G, x_i \vDash \psi$ for $1 \leq i \leq n$. Again, by Lemma 3.7 we know there is a set of strings U equivalent to E in both G and G'. By Lemma 5.9 there are only 10 types of strings. By induction, we consider three cases.

   First, if $[\![\psi]\!]^G \cap V = A \cup B$, then $x_i, \dots, x_n \in A \cup B$. Therefore, we know U must at least contain strings of type 6 or 9. Suppose U contains strings of type 6. Then, we verify that $\sharp(\bigcup_{s\in U}[\![s]\!]^G(b) \cap (A \cup B)) \geq m \geq n$. Otherwise, whenever U contains strings of type 9, and not of type 6, we know $n = 1$ and clearly $\sharp(\bigcup_{s\in U}[\![s]\!]^G(b) \cap (A \cup B)) = 1$.

Next, if $[\![\psi]\!]^G \cap V = C$, then $x_i, \ldots, x_n \in C$. Therefore, we know U must at least contain strings of type 3, 4 or 8. Whenever U contains 3, 4 or 8, we verify that $\sharp(\bigcup_{s \in U} [\![s]\!]^G(b) \cap C) \geq \frac{m}{2} \geq n$.

Next, if $[\![\psi]\!]^G \cap V = V$, then $x_i, \ldots, x_n \in V$. Therefore, we know U must at least contain strings of type 3, 4, 6, 8 or 9. All these types have already been handled in the previous two cases.

Finally, the case where $[\![\psi]\!]^G \cap V = \emptyset$ cannot occur as $n > 0$.

The sets of strings U above are also exactly the sets used to argue the implication from right to left.

(ii) For every case of U above, for every inductive case of ψ, we can also verify that $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(a) \cap (A \cup B)) \geq m \geq n$, $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(a) \cap C) \geq \frac{m}{2} \geq n$, and $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(a) \cap V) \geq \frac{m}{2} \geq n$.

(iii) Assume $G, c \vDash \phi$. Then, there exist distinct $x_1, \ldots, x_n$ s.t. $(c, x_i) \in [\![E]\!]^G$ and $G, x_i \vDash \psi$ for $1 \leq i \leq n$. By induction, we consider three cases.

First, if $[\![\psi]\!]^G \cap V = A \cup B$, then $x_i, \ldots, x_n \in A \cup B$. Therefore, we know U must at least contain strings of type 1, 2 or 7. Whenever U contains 1, 2 or 7, we verify that $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(c) \cap (A \cup B)) \geq \frac{m}{2} \geq n$.

Next, if $[\![\psi]\!]^G \cap V = C$, then $x_i, \ldots, x_n \in C$. Therefore, we know U must at least contain strings of type 5 or 9. Whenever U contains 5, we verify that $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(c) \cap C) \geq m \geq n$. Otherwise, whenever U contains strings of type 9, and not of type 5, we know $n = 1$ and clearly $\sharp(\bigcup_{s \in U} [\![s]\!]^{G'}(c) \cap C) = 1$.

Next, if $[\![\psi]\!]^G \cap V = V$, then $x_i, \ldots, x_n \in V$. Therefore, we know U must at least contain strings of type 1, 2, 5, 7 or 9. All these types have already been handled in the previous two cases.

Finally, the case where $[\![\psi]\!]^G \cap V = \emptyset$ cannot occur as $n > 0$.

The sets of strings U above are also exactly the sets used to argue the implication from right to left.

(iv) For the direction from left to right, take $x \in [\![\phi]\!]^G \setminus V$. Since $G, x \vDash \phi$, there exists $y_1, \ldots, y_n$ s.t. $(x, y_i) \in [\![E]\!]^G$ and $G, y_i \vDash \psi$ for $i = 1, \ldots, n$. However, since $x \notin V$, by Lemma 3.5, all y_i must equal x. Hence, $n = 1$ and $(x, x) \in [\![E]\!]^G$ and $G, x \vDash \psi$. Then again, by the same Lemma, $(x, x) \in [\![E]\!]^{G'}$, since G and G' have the same set of nodes V/ Moreover, by induction, $G', x \vDash \psi$. We conclude $G', x \vDash \phi$ as desired. The direction from right to left is argued symmetrically. □

Remark 5.11. In our construction of the graphs G and G', we work with segments that overlap for 1/8th of the number of nodes. The critical reader will remark that an overlap of a single node would already be sufficient. Our choice for working with a larger overlap is indeed largely aesthetic. Moreover, our proof still works for an extension of SHACL where shapes of the form $|r \cap E| \geq n$ would be allowed. This extension allows us to write shapes like $|\mathsf{colleague} \cap \mathsf{friend}| \geq 5$, stating that the node has at least five colleagues that are also friends. Such an extension then would still not be able to express full disjointness.

5.3. **Further non-definability results.** In Theorem 3.1, we showed that equality is primitive in $\mathcal{L}(\textit{disj}, \textit{closed})$, and similarly, that disjointness is primitive in $\mathcal{L}(\textit{eq}, \textit{closed})$. Can we strengthen these results to $\mathcal{L}(\textit{full-disj}, \textit{closed})$ and $\mathcal{L}(\textit{full-eq}, \textit{closed})$, respectively? This turns out to be indeed possible.

Table 3: Sets of types starting from a_i, b_i in G and a_i in G'.

U	$[\![E]\!]^G(a_i)$	$[\![E]\!]^G(b_i)$	$[\![E]\!]^{G'}(a_i)$
$\{\mathbf{1}\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\{2\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\{\mathbf{3}\}$	$c_{i-\frac{k}{2}+1\to i}$	$c_{i-\frac{k}{2}+1\to i+\frac{k}{8}}$	$c_{i-\frac{k}{2}+1\to i+\frac{k}{8}}$
$\{\mathbf{4}\}$	$c_{i+1\to i+\frac{k}{2}}$	$c_{i-\frac{k}{8}+1\to i+\frac{k}{2}}$	$c_{i-\frac{k}{8}+1\to i+\frac{k}{2}}$
$\{5\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\{\mathbf{6}\}$	$A \cup B$	$A \cup B$	$A \cup B$
$\{7\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\{\mathbf{8}\}$	C	C	C
$\{\mathbf{9}\}$	$\{a_i\}$	$\{b_i\}$	$\{a_i\}$
$\{10\}$	$\emptyset$	$\emptyset$	$\emptyset$
$\{3,4\}$	C	C	C
$\{\mathbf{3},\mathbf{6}\}$	$A \cup B \cup c_{i-\frac{k}{2}+1\to i}$	$A \cup B \cup c_{i-\frac{k}{2}+1\to i+\frac{k}{8}}$	$A \cup B \cup c_{i-\frac{k}{2}+1\to i+\frac{k}{8}}$
$\{3,8\}$	C	C	
$\{\mathbf{3},\mathbf{9}\}$	$c_{i+1\to i+\frac{k}{2}} \cup \{a_i\}$	$c_{i-\frac{k}{8}+1\to i+\frac{k}{2}} \cup \{b_i\}$	$c_{i-\frac{k}{8}+1\to i+\frac{k}{2}} \cup \{a_i\}$
$\{\mathbf{4},\mathbf{6}\}$	$A \cup B \cup c_{i+1\to i+\frac{k}{2}}$	$A \cup B \cup c_{i-\frac{k}{8}+1\to i+\frac{k}{2}}$	$A \cup B \cup c_{i-\frac{k}{8}+1\to i+\frac{k}{2}}$
$\{4,8\}$	C	C	
$\{\mathbf{4},\mathbf{9}\}$	$c_{i+1\to i+\frac{k}{2}} \cup \{a_i\}$	$c_{i-\frac{k}{8}+1\to i+\frac{k}{2}} \cup \{b_i\}$	$c_{i-\frac{k}{8}+1\to i+\frac{k}{2}} \cup \{a_i\}$
$\{\mathbf{6},\mathbf{8}\}$	V	V	V
$\{6,9\}$	$A \cup B$	$A \cup B$	$A \cup B$
$\{\mathbf{8},\mathbf{9}\}$	$C \cup \{a_i\}$	$C \cup \{b_i\}$	$C \cup \{a_i\}$
$\{3,6,4\}$	V	V	V
$\{3,6,8\}$	V	V	V
$\{3,6,9\}$	$A \cup B \cup c_{i-\frac{k}{2}+1\to i}$	$A \cup B \cup c_{i-\frac{k}{2}+1\to i+\frac{k}{8}}$	$A \cup B \cup c_{i-\frac{k}{2}+1\to i+\frac{k}{8}}$
$\{3,9,4\}$	$C \cup \{a_i\}$	$C \cup \{b_i\}$	$C \cup \{a_i\}$
$\{4,9,6\}$	$A \cup B \cup c_{i+1\to i+\frac{k}{2}}$	$A \cup B \cup c_{i-\frac{k}{8}+1\to i+\frac{k}{2}}$	$A \cup B \cup c_{i-\frac{k}{8}+1\to i+\frac{k}{2}}$
$\{4,9,8\}$	C	C	C
$\{8,9,3\}$	$C \cup \{a_i\}$	$C \cup \{b_i\}$	$C \cup \{a_i\}$
$\{4,6,3\}$	V	V	V

That equality remains primitive in $\mathcal{L}(\textit{full-disj}, \textit{closed})$ already follows from our given proof in Section 3.3. In effect, the attentive reader may have noticed that we already cover full disjointness in that proof. In contrast, our proof of primitivity of disjointness in $\mathcal{L}(\textit{eq}, \textit{closed})$ does not extend to full equality. Nevertheless, we can reuse our proof of primitivity of *full* disjointness as follows. The graphs G and G' from Proposition 5.10 are indistinguishable in $\mathcal{L}(\textit{full-eq}, \textit{disj}, \textit{closed})$. Let H and H' be the same graphs but with all directed edges reversed (i.e., the graphs illustrated in Figure 4). Then the same proof shows that H and H' are indistinguishable in $\mathcal{L}(\textit{full-eq}, \textit{closed})$. However, since G and G' are distinguishable by the inclusion statement $\exists p^-.\top \subseteq \neg \textit{disj}(p^-, q^-)$, also H and H' are distinguishable by the inclusion statement $\exists p.\top \subseteq \neg \textit{disj}(p, q)$. Thus, the primitivity of disjointness in $\mathcal{L}(\textit{full-eq}, \textit{closed})$ is established.

Table 4: Sets of types starting from c_i in G and in G'.

U	$[\![E]\!]^G(c_i)$	$[\![E]\!]^{G'}(c_i)$
$\{\mathbf{1}\}$	$a_{i\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1}$	$a_{i-\frac{m}{8}\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1}$
$\{\mathbf{2}\}$	$a_{i-\frac{m}{2}\to i-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1}$	$a_{i-\frac{m}{2}\to i+\frac{m}{8}-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1}$
$\{\mathbf{3}\}$	$\emptyset$	$\emptyset$
$\{4\}$	$\emptyset$	$\emptyset$
$\{\mathbf{5}\}$	C	C
$\{6\}$	$\emptyset$	$\emptyset$
$\{\mathbf{7}\}$	$A \cup B$	$A \cup B$
$\{8\}$	$\emptyset$	$\emptyset$
$\{\mathbf{9}\}$	$\{c_i\}$	$\{c_i\}$
$\{10\}$	$\emptyset$	$\emptyset$
$\{1, 2\}$	$A \cup B$	$A \cup B$
$\{\mathbf{1}, \mathbf{5}\}$	$C \cup a_{i\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1}$	$C \cup a_{i-\frac{m}{8}\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1}$
$\{1, 7\}$	$A \cup B$	$A \cup B$
$\{\mathbf{1}, \mathbf{9}\}$	$\{c_i\} \cup a_{i\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1}$	$\{c_i\} \cup a_{i-\frac{m}{8}\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1}$
$\{\mathbf{2}, \mathbf{5}\}$	$C \cup a_{i-\frac{m}{2}\to i-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1}$	$C \cup a_{i-\frac{m}{2}\to i+\frac{m}{8}-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1}$
$\{2, 7\}$	$A \cup B$	$A \cup B$
$\{\mathbf{2}, \mathbf{9}\}$	$\{c_i\} \cup a_{i-\frac{m}{2}\to i-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1}$	$\{c_i\} \cup a_{i-\frac{m}{2}\to i+\frac{m}{8}-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1}$
$\{\mathbf{5}, \mathbf{7}\}$	V	V
$\{5, 9\}$	C	C
$\{\mathbf{7}, \mathbf{9}\}$	$\{c_i\} \cup A \cup B$	$\{c_i\} \cup A \cup B$
$\{1, 5, 2\}$	V	V
$\{1, 5, 7\}$	V	V
$\{1, 5, 9\}$	$C \cup a_{i\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1}$	$C \cup a_{i-\frac{m}{8}\to i+\frac{m}{2}-1} \cup b_{i-\frac{m}{8}\to i+\frac{m}{2}-1}$
$\{1, 9, 2\}$	$\{c_i\} \cup A \cup B$	$\{c_i\} \cup A \cup B$
$\{1, 9, 7\}$	$\{c_i\} \cup A \cup B$	$\{c_i\} \cup A \cup B$
$\{2, 5, 7\}$	V	V
$\{2, 5, 9\}$	$C \cup a_{i-\frac{m}{2}\to i-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1}$	$C \cup a_{i-\frac{m}{2}\to i+\frac{m}{8}-1} \cup b_{i-\frac{m}{2}\to i+\frac{m}{8}-1}$
$\{2, 9, 7\}$	$\{c_i\} \cup A \cup B$	$\{c_i\} \cup A \cup B$
$\{5, 7, 9\}$	V	V

6. 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 6.1. 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 (possibly empty) 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 $[\![s]\!]^I$ 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 $[\![\phi]\!]^J$ is a subset of $[\![s]\!]^J$. 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 $\mathcal{P}$ (Indeed, J is the least fixpoint of the well-known immediate consequence operator, which is a monotone operator since $\mathcal{P}$ is semipositive [AHV95]). 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 $\mathcal{P}_1, \ldots, \mathcal{P}_n$ 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 $\mathcal{P}_1, \ldots, \mathcal{P}_n$ 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 $\mathcal{P}$.

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

Remark 6.2. The nonrecursive notion of shape schema, defined in Section 2, corresponds to the special case where $\mathcal{P}$ is the empty program.

Extending Theorem 3.1. Theorem 3.1 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 $[\![s]\!]^{\mathcal{P}(H)} = [\![\phi]\!]^H$. 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 3.2 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 3.2, which establishes the theorem for features X other than *closed*.

Also for $X = \mathit{closed}$, the reasoning, given after Lemma 3.16, 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 4.1. Also Theorem 4.1 extends to stratified shape schemas. Thereto, Lemma 3.16 needs to be reproven in the presence of a stratified program $\mathcal{P}$ defining the intensional shape names. The extended Lemma 3.16 then states that $[\![\phi]\!]^{\mathcal{P}(G)} = [\![\phi]\!]^{\mathcal{P}(G')}$. The proof of Theorem 4.1 then goes through unchanged.

Extending Theorem 5.2. Also Theorem 5.2 extends to stratified shape schemas for the same reasons given above for Theorem 3.1.

7. 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 [AHV95, Tru18]. 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. [ACO$^+$20] 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 [BGP17]. Unified approaches developed for logic programming semantics can be naturally applied to SHACL [BJ21].

Notably, Corman et al. [CRS18] have already suggested that disjointness is redundant in a setting of recursive shape schemas with nonstratified negation. Their expression is not correct, however [Reu21].[5]

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 [CFRS19, T$^+$10, DMH$^+$21]. Certainly, the affinity with description logics makes it easy to carve out cases where higher reasoning tasks become decidable [LS$^+$20, PK$^+$20]. 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 [Var82] compare?

[5]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.

Acknowledgement

We thank the anonymous referees for helpful comments that improved the presentation of this paper. This research was supported by the Flanders AI Research programme.

References

[ACO+20] 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. doi:10.1145/3366423.3380229.

[AGSS86] 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.

[AHV95] S. Abiteboul, R. Hull, and V. Vianu. *Foundations of Databases*. Addison-Wesley, 1995.

[BGP17] 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. doi:10.1007/978-3-319-68288-4_7.

[BHLS17] F. Baader, I. Horrocks, C. Lutz, and U. Sattler. *An Introduction to Description Logic*. Cambridge University Press, 2017.

[BJ21] B. Bogaerts and M. Jakubowski. Fixpoint semantics for recursive SHACL. In *Proceedings 37th International Conference on Logic Programming (Technical Communications), ICLP Technical Communications 2021, Porto (virtual event), 20-27th September 2021*, volume 345, pages 41–47, 2021. doi:10.4204/EPTCS.345.14.

[BJVdB22] B. Bogaerts, M. Jakubowski, and J. Van den Bussche. Expressiveness of SHACL features. In Dan Olteanu and Nils Vortmeier, editors, *Proceedings of ICDT*, volume 220, pages 15:1–15:16. Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 2022. doi:10.4230/LIPIcs.ICDT.2022.15.

[CDGNL03] D. Calvanese, G. De Giacomo, D. Nardi, and M. Lenzerini. Reasoning in expressive description logics. In F. Baader, D. Calvanese, D. McGuiness, D. Nardi, and P. Patel-Schneider, editors, *The Description Logic Handbook*, chapter 23. Cambridge University Press, 2003.

[CFRS19] 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. doi:10.1007/978-3-030-30793-6_9.

[CRS18] J. Corman, J.L. Reutter, and O. Savkovic. Semantics and validation of recursive SHACL. In D. Vrandecic 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/krdb/tech-reports/.

[DMH+21] B. De Meester, P. Heyvaert, et al. RDF graph validation using rule-based reasoning. *Semantic Web*, 12(1):117–142, 2021.

[HS94] R. Hull and J. Su. Domain independence and the relational calculus. *Acta Informatica*, 31:513–524, 1994. doi:10.1007/BF01213204.

[Jak22] M. Jakubowski. Extending SHACL, 2022. URL: https://www.mjakubowski.info/posts/datashapes.html.

[Knu21] H. Knublauch. Dash constraint components. https://datashapes.org/constraints.html, 2021.

[LS+20] M. Leinberger, P. Seifer, et al. Deciding SHACL shape containment through description logics reasoning. In Pan et al. [P+20], pages 366–383. doi:10.1007/978-3-030-62419-4_21.

[P+20] J.Z. Pan et al., editors. *Proceedings 19th International Semantic Web Conference*, volume 12506 of *Lecture Notes in Computer Science*. Springer, 2020.

[PK+20] P. Pareti, G. Konstantinidis, et al. SHACL satisfiability and containment. In Pan et al. [P+20], pages 474–493. doi:10.1007/978-3-030-62419-4_27.

[RDF14] RDF 1.1 primer. W3C Working Group Note, June 2014.

[Reu21] J. Reutter. Personal communication, 15 January 2021.

[SHA17] Shapes constraint language (SHACL). W3C Recommendation, July 2017.

[T+10] J. Tao et al. Integrity constraints in OWL. In *Proceedings 24th AAAI Conference on Artificial Intelligence*, pages 1443–1448, 2010. doi:10.5555/2898607.2898837.

[Tru18] M. Truszczynski. 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.

[Var82] M. Vardi. The complexity of relational query languages. In *Proceedings 14th ACM Symposium on the Theory of Computing*, pages 137–146, 1982.

Appendix A. Supplementary Proofs

Proof of Lemma 3.7. We first state an auxiliary lemma:

Lemma A.1. *Let V be a finite set of n elements, and let $R \subseteq V \times V$ be a binary relation over V. We have $R^* = R^0 \cup R^1 \cup \cdots \cup R^{n-1}$.*

Proof. R^* is defined as $R^0 \cup R^1 \cup \dots$ however, we will show that if $(a, b) \in R^m$, with $m \geq n$, then there exists a $k < m$ such that $(a, b) \in R^k$.

We call a sequence of elements $x_1, \dots, x_h$ an R-path if $(x_l, x_{l+1}) \in R$ for $1 \leq l \leq h$.

If $(a, b) \in R^m$, then there exists an R-path $x_1, \dots, x_{m+1}$ with $x_1 = a$ and $x_{m+1} = b$. As there are only n total elements, there exists i, j with $1 \leq i < j \leq m + 1$ such that $x_i = x_j$. Therefore, $x_1, \dots, x_{i-1}, x_j, \dots, x_{m+1}$ is also an R-path. We conclude that $(a, b) \in R^{m-(j-i)}$, as desired. □

Proof of Lemma 3.7. The proof is by induction on the structure of E. Clearly for the base case $E = p$, we have the set $U = \{p\}$ and similarly for $E = p^-$ we have $U = \{p^-\}$. When $E = id$, clearly $U = \{id\}$. Next, we consider the inductive cases. When $E = E_1 \cup E_2$, we know by induction there exists a set of strings U_1 for E_1, and U_2 for E_2. We then have $U = U_1 \cup U_2$. When $E = E_1/E_2$, we again know by induction there exists a set of strings U_1 for E_1, and U_2 for E_2. We have $U = \{s_1/s_2 \mid s_1 \in U_1 \text{ and } s_2 \in U_2\}$. Finally, when $E = E'^*$, we know by induction there exists a set of strings U' for E'. Let W be a set of strings, we define $W^1 := W$, and for a natural number $m > 1$, $W^m := \{s_1/s_2 \mid s_1 \in W, s_2 \in W^{m-1}\}$. We also use the shorthand notation E^m, with $m > 0$ a natural number, to denote m compositions of the path expression E. For example, E^3 is $E/E/E$. By definition, $[\![E'^*]\!]^G = [\![id]\!]^G \cup [\![E']\!]^G \cup [\![E'^2]\!]^G \cup \dots$. By Lemma A.1 we know that this is the same as $[\![E'^*]\!]^G = [\![id]\!]^G \cup [\![E']\!]^G \cup [\![E'^2]\!]^G \cup \cdots \cup [\![E'^{n-1}]\!]^G$ for graphs with at most n nodes. It then follows that $U = \{id\} \cup U' \cup U'^2 \cup \cdots \cup U'^{n-1}$. □

Proof of Lemma 3.11.

Proof. For $i = 1, 2, 3, 4$, define the *i-th blob of nodes* to be the set $X_i = \{x_i^1, \dots, x_i^M\}$ (see Figure 2). 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^-, the first claim is clear because $[\![r^-]\!]^{G'} \subseteq [\![E]\!]^{G'}$, and we only need to verify the second one. That holds because for any i, if $v \in X_i$, then $[\![p^-]\!]^{G'}(v) \supseteq X_{prev(i)}$ and clearly $X_{prev(i)} - [\![r]\!]^{G'}(v) \neq \emptyset$. 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 two claims immediately follow by induction, since $[\![E]\!]^{G'} \supseteq [\![E_1]\!]^{G'}$ and $[\![E]\!]^{G'} \supseteq [\![E_2]\!]^{G'}$. 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^*. If E_1 is not simple, the two claims follow immediately by induction, since $[\![E]\!]^{G'} \supseteq [\![E_1]\!]^{G'}$. If E_1 is simple, the first claim clearly hold for E, so we only need to verify the second claim. That holds because, by the form of E, every node v is in $[\![E]\!]^{G'}(v)$, but not in $[\![r]\!]^{G'}(v)$, 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 claim one holds because $[\![r]\!]^{G'} \subseteq [\![E]\!]^{G'}$. 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 second statement to E_1 and E_2.

We distinguish two cases.

- If $[\![r]\!]^{G'} \subseteq [\![E_2]\!]^{G'}$, then we show that $[\![r]\!]^{G'} \subseteq [\![E]\!]^{G'}$. Let $v \in X_i$. We verify the following two inclusions:
  - $[\![E]\!]^{G}(v) \supseteq X_i$. Let $u \in X_i$. If $u \neq v$, choose a third node $w \in X_i$. Since X_i is a clique, $(v, w) \in [\![E_1]\!]^{G}$ because the first claim holds for E_1. By $[\![r]\!]^{G'} \subseteq [\![E_2]\!]^{G'}$, we also have $(w, u) \in [\![E_2]\!]^{G'}$, whence $u \in [\![E]\!]^{G'}(v)$ as desired. If $u = v$, we similarly have $(v, w) \in [\![E_1]\!]^{G'}$ and $(w, u) \in [\![E_2]\!]^{G'}$ as desired.
  - $[\![E]\!]^{G}(v) \supseteq X_{next(i)}$. Let $u \in X_{next(i)}$ and choose $w \neq v \in X_i$. Because the first claim holds for E_1, we have $(v, w) \in [\![E_1]\!]^{G}$. By $[\![r]\!]^{G'} \subseteq [\![E_2]\!]^{G'}$, we also have $(w, u) \in [\![E_2]\!]^{G'}$, whence $u \in [\![E]\!]^{G'}(v)$ as desired.

  We conclude that $[\![E]\!]^{G'}(v) \supseteq X_i \cup X_{next(i)} \supseteq [\![r]\!]^{G'}$ as desired.
- If $[\![r^-]\!]^{G'} \subseteq [\![E_2]\!]^{G'}$, then we show that $[\![r^-]\!]^{G'} \subseteq [\![E]\!]^{G'}$. This is analogous to the previous case, now verifying that $[\![E]\!]^{G}(v) \supseteq X_i \cup X_{prev(i)}$.

In both cases, the second statement now follows for every node v. Indeed, $v \in X_i \subseteq [\![E]\!]^{G'}(v)$ but $v \notin [\![r]\!]^{G'}(v)$. ☐