Common Foundations for Recursive Shape Languages

Shqiponja Ahmetaj[1], **Iovka Boneva**[2], **Jan Hidders**[3], **Maxime Jakubowski**[1], **Jose-Emilio Labra-Gayo**[4], **Wim Martens**[5], **Fabio Mogavero**[6], **Filip Murlak**[7], **Cem Okulmus**[8], **Ognjen Savković**[9], **Mantas Šimkus**[1], **Dominik Tomaszuk**[10]

[1]TU Wien, Austria
[2]Univ. Lille, F-59000 Lille, France
[3]Birkbeck, University of London, UK
[4]University of Oviedo, Spain
[5]University of Bayreuth, Germany
[6]Universita di Napoli Federico II, Italy
[7]University of Warsaw, Poland
[8]Paderborn University, Germany
[9]Free University of Bolzano, Italy
[10]University of Białystok, Poland

Abstract

As schema languages for RDF data become more mature, we are seeing efforts to extend them with recursive semantics, applying diverse ideas from logic programming and description logics. While ShEx has an official recursive semantics based on *greatest fixpoints* (GFP), the discussion for SHACL is ongoing and seems to be converging towards *least fixpoints* (LFP). A practical study we perform shows that, indeed, ShEx validators implement GFP, whereas SHACL validators are more heterogeneous. This situation creates tension between ShEx and SHACL, as their semantic commitments appear to diverge, potentially undermining interoperability and predictability. We aim to clarify this design space by comparing the main semantic options in a principled yet accessible way, hoping to engage both theoreticians and practitioners, especially those involved in developing tools and standards. We present a unifying formal semantics that treats LFP, GFP, and supported model semantics (SMS), clarifying their relationships and highlighting a duality between LFP and GFP on stratified fragments. Next, we investigate to which extent the directions taken by SHACL and ShEx are compatible. We show that, although ShEx and SHACL seem to be going in different directions, they include large fragments with identical expressive power. Moreover, there is a strong correspondence between these fragments through the aforementioned principle of duality. Finally, we present a complete picture of the data and combined complexity of ShEx and SHACL validation under LFP, GFP, and SMS, showing that SMS comes at a higher computational cost under standard complexity-theoretic assumptions.

1 Introduction

Graph-based data representations are rapidly gaining importance due to the unprecedented growth of interconnected data and their increasing role in AI-driven systems (Sakr et al. 2021). Furthermore, their flexibility makes them an ideal tool for bridging the gap between structured data and machine learning tools in industrial applications (Anuradha et al. 2023; Mohamed et al. 2022). Since automatic processing of data (graph or otherwise) is significantly facilitated by the presence of a *schema*, we have seen various efforts to design schema languages for graph-based data representations, notably ShEx (Prud'hommeaux, Labra Gayo, and Solbrig 2014; Staworko et al. 2015; Prud'hommeaux et al. 2019), SHACL (Knublauch and Kontokostas 2017), and PG-Schema (Angles et al. 2021; Angles et al. 2023). In this paper, we zoom in on *recursion* as a foundational mechanism that directly shapes the semantics of cyclic schema dependencies. We therefore focus on ShEx and SHACL, the de facto standard schema languages for RDF, since PG-Schema does not use this kind of recursion.

It is well-known that designing declarative languages with recursion and negation is non-trivial (Abiteboul, Hull, and Vianu 1995; Aref et al. 2025). SHACL and ShEx's open-world nature (contrasting the typical database setting) does not make this task any easier. This might explain why the design of recursion is taking different directions in ShEx and SHACL. While ShEx has long had an official semantics based on *greatest fixpoints* (GFP) (Prud'hommeaux et al. 2019; Staworko et al. 2015; Boneva, Labra Gayo, and Prud'hommeaux 2017), the discussion for SHACL is ongoing. Proposals include a *supported-model semantics* (SMS) (Corman, Reutter, and Savković 2018), grounded in first-order logic, as well as alternatives inspired by fixpoint and logic-programming paradigms (Andresel et al. 2020; Okulmus and Šimkus 2024; Bogaerts and Jakubowski 2021; Ahmetaj et al. 2022b; Pareti, Konstantinidis, and Mogavero 2022). The academic discussion appears to be converging towards *least fixpoints* (LFP), but the issue remains to be addressed by the W3C working group.

The current situation causes some tension between ShEx and SHACL since the proposals seem to diverge, which may cause interoperability issues in the future. Interoperability is important in large-scale industry applications. ShEx and SHACL can happily co-exist and have their own approach and identity, but the more *fundamentally different* they are, the more painful it becomes to switch from one to the other

if we realise at a late development stage that we have chosen the wrong language.

Our Contributions. (1) We shed light on this matter by proposing a unifying formal framework that is compatible with the work of Ahmetaj et al. (Ahmetaj et al. 2025a). We define a *simple shape language* (SSL) with the three main options for recursion: LFP, GFP, and SMS (Section 2). The language SSL is carefully designed to be much simpler than ShEx and SHACL, yet sufficiently powerful to illustrate the fundamental tradeoffs between LFP, GFP, and SMS, and to test which semantics is used by validation engines. Despite its simplicity, it is still expressive: it corresponds to the alternation-free fragment of the modal μ-calculus (Kozen 1983) with nominals but without atomic propositions, which is an important yardstick. Using the duality principle of fixpoint theory, we observe that SSL with LFP is equally expressive as SSL with GFP, using a straightforward translation. This close relationship is good news for the compatibility between ShEx with GFP and SHACL with LFP. The situation for SMS differs, see (4).

(2) We experimentally explore whether existing SHACL and ShEx validators use LFP, GFP, or SMS (Section 3). Our tests reveal that current tools implicitly commit to different semantics, which leads to non-uniform behaviour across implementations. In a nutshell, ShEx validators seem to implement GFP, whereas SHACL validators are more heterogeneous. This study is an important motivation of our work. It confirms our hypothesis that validation engines are not aligned and underscores why there is a need for a consensus on recursive semantics in SHACL. Once a semantics for recursion in SHACL is agreed upon, our tests can be added to the official test suite (Labra Gayo, Knublauch, and Kontokostas 2024).

(3) We then move to the formal study of real schema languages. Though neat and relatively expressive, SSL is still a small part of real ShEx and SHACL. In Sections 4.1–4.2, we discuss translations between large fragments of ShEx (with GFP) and SHACL with LFP. Our translations show that these larger fragments have the same expressive power and that the duality principle that we saw for SSL extends to real-world languages.

(4) We provide a complete overview of the data and combined complexity of ShEx, SHACL, and SSL with LFP, GFP, and SMS (Section 4.3). In some cases the complexity has been known already (Andresel et al. 2020; Staworko et al. 2015; Bogaerts, Jakubowski, and Van den Bussche 2024; Bogaerts and Jakubowski 2021); we complete the picture by establishing tight bounds for the combined and data complexity for ShEx. The main conclusion from the complete set of results is that LFP and GFP allow polynomial-time data complexity for all languages, whereas SMS immediately leads to intractability, even for very simple languages like SSL. This means that LFP and GFP are much easier to deal with from an algorithmic perspective, and would be our recommendation for recursion in ShEx and/or SHACL.

(5) As a group of authors including contributors to the design of both ShEx and SHACL, we reach a common understanding of the consequences of the choice of the semantics for recursion, and clarify that it is fine for SHACL to use LFP, while ShEx uses GFP. In a nutshell, LFP is a very sensible choice because it is natural, its expressiveness is compatible with ShEx's GFP (allowing effective translation), and has good complexity properties (see also Section 5).

Missing proofs can be found in the full version of this paper (Ahmetaj et al. 2026).

Related Work. The development of constraint languages for graph-based data has a rich history. SHACL (Knublauch and Kontokostas 2017) was designed taking the initial inspiration from SPIN Rules (Knublauch, Hendler, and Idehen 2011), whereas ShEx (Prud'hommeaux, Labra Gayo, and Solbrig 2014) draws more heavily on paradigms established in XML Schema and RELAX NG (Gao et al. 2012; Bray et al. 2008; Martens et al. 2017). Both languages aim to enable *prescriptive* validation of RDF, in contrast to the more *descriptive* nature of languages such as OWL (W3C OWL Working Group 2012). Nevertheless, they diverge in several key features, which poses challenges for tool interoperability and for users seeking to transition between the two frameworks. In our prior work (Ahmetaj et al. 2025a), we took initial steps toward reconciling these differences by proposing a common framework for ShEx, SHACL, and PG-Schema, which was developed to provide similar schema capabilities in the context of the recent GQL standard (ISO/IEC 39075 2024; Deutsch et al. 2022). This work builds on that foundation by focusing on recursion, which the earlier study excluded.

Before this, expressiveness and computational properties of (non-recursive) SHACL have been extensively studied (Bogaerts, Jakubowski, and Van den Bussche 2022; Bogaerts, Jakubowski, and Van den Bussche 2024; Pareti et al. 2020; Pareti, Konstantinidis, and Mogavero 2022). Similarly, the foundations of ShEx, including its formal semantics and complexity, have been explored in depth (Staworko et al. 2015; Boneva, Labra Gayo, and Prud'hommeaux 2017).

Recursion in Shape Languages. The semantic challenges of recursion for shape languages, and particularly its interaction with negation, have been a significant topic of recent research. This is especially pronounced for SHACL, where the W3C recommendation (Knublauch and Kontokostas 2017) notably left the semantics of recursion undefined. The challenges of recursion were first formally addressed by Corman et al. (Corman, Reutter, and Savković 2018; Corman et al. 2019), who proposed a *supported model semantics* based on first-order logic. Subsequent work has explored various semantics mirroring established paradigms from fixpoint logics and logic programming (Andresel et al. 2020; Okulmus and Šimkus 2024; Bogaerts and Jakubowski 2021; Ahmetaj et al. 2022b; Pareti, Konstantinidis, and Mogavero 2022). In contrast to SHACL, the semantics of recursion in ShEx has been consistently defined via *greatest fixpoints* (Boneva, Labra Gayo, and Prud'hommeaux 2017; Staworko et al. 2015). This divergence in the formalization process lead to significant inconsistencies across validators.

Validation in Practice. There is a growing body of work on the practical use of shape languages. This includes mining shapes from large knowledge graphs (Rabbani, Lissandrini, and Hose 2023; Lissandrini, Rabbani, and

Hose 2024), validating real-world datasets such as Wikidata (Ferranti et al. 2024), and studying data provenance and explanations for validation outcomes (Delva et al. 2023; Ahmetaj et al. 2022a). The need to validate heterogeneous graph data has also motivated comparative studies of SHACL and ShEx (Labra Gayo et al. 2017; Labra Gayo et al. 2019; Tomaszuk 2017) as well as mappings between RDF and Property Graph paradigms (Angles, Thakkar, and Tomaszuk 2020; Hartig 2014). Further extensions of ShEx for different graph types have been proposed (Labra Gayo 2024; Labra Gayo 2022), alongside unifying graph data models such as OneGraph (Lassila et al. 2023) and MillenniumDB's Domain Graph Model (Vrgoč et al. 2023). Semantic ambiguities in recursive SHACL directly affect these practical efforts, since the outcome of validation can depend on the choice of validator and its implicit semantics.

Connections to Description Logics and μ-calculus. Foundational work on SHACL has revealed strong connections to expressive Description Logics (DLs), the logics underlying OWL (Bogaerts, Jakubowski, and Van den Bussche 2022; Leinberger et al. 2020; Ahmetaj et al. 2021). The problem of defining coherent semantics for recursive shape languages closely parallels the classic issue of *terminological cycles* in DLs, extensively studied in the early 1990s. Terminologies with recursive definitions, where concept names may be defined directly or indirectly in terms of themselves, were recognized to require special semantic treatment to ensure meaningful interpretations. Initial studies (Baader 1990; Baader 1996; Nebel 1990) identified three main approaches to interpreting such cycles: least fixpoint, greatest fixpoint, and descriptive (classical) semantics. It was observed that, for many cyclic definitions, the greatest-fixpoint semantics yields more satisfactory results by allowing maximal solutions consistent with their intended meaning, whereas for others, the least-fixpoint semantics is more suitable (Baader 1996; De Giacomo and Lenzerini 1994). Further work argued that, instead of selecting a single semantics, better results are obtained by adopting a formalism that allows for the different semantics to coexist, where the correspondence with the modal μ-calculus was also established (Schild 1994; De Giacomo and Lenzerini 1994). In our setting, we show that expressive yet tractable fragments of recursive SHACL and ShEx correspond to the alternation-free fragment of the modal μ-calculus (Kozen 1983). We finally remark that the recent work in (Oudshoorn, Ortiz, and Šimkus 2026) uses μ-calculus to show decidability and complexity results for static analysis tasks in SHACL under the *well-founded semantics*.

2 The Multiple Semantics of Recursion

We now introduce the *simple shape language*, which is a core of both ShEx and SHACL, and formally define the three main modes of recursion that are being considered.

2.1 The RDF Data Model

An RDF graph is a finite set of triples in $(\mathsf{IRIs} \cup \mathsf{Blanks}) \times (\mathsf{IRIs}) \times (\mathsf{IRIs} \cup \mathsf{Blanks} \cup \mathsf{Literals})$, where IRIs, Literals and Blanks are three countable sets of IRIs, literal data values, and blank nodes, respectively. If (u, p, v) is a triple of an RDF graph, then u is called its subject, p its predicate, and v its object. The set of nodes of an RDF graph $\mathcal{G}$, denoted $\mathsf{Nodes}(\mathcal{G})$, is the set of IRIs, literals or blank nodes that appear in a subject or an object position in some triple of $\mathcal{G}$. We sometimes use *predicates* to refer to the elements of IRIs when they are used in a predicate position of some triple.

To illustrate, consider the RDF graph $\mathcal{G}$:

$$\mathcal{G} = \left\{ \begin{matrix} (\texttt{ex:Alice}, \texttt{ex:knows}, \texttt{ex:Bob}), \\ (\texttt{ex:Bob}, \texttt{ex:age}, 42) \end{matrix} \right\}.$$

Here `ex:Alice` and `ex:Bob` are IRIs used as nodes, `ex:knows` and `ex:age` are IRIs used as predicates, and 42 is a literal value. In this case $\mathsf{Nodes}(\mathcal{G}) = \{\texttt{ex:Alice}, \texttt{ex:Bob}, 42\}$.

φ'	$[\![\varphi']\!]^{\alpha}_{\mathcal{G},\mathcal{C}}$
$\bot$	$\emptyset$
$\top$	$\mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C})$
$\mathsf{test}(c)$	$\{c\}$
s	$\alpha(s)$
$\neg\varphi$	$(\mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C})) \setminus [\![\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}$
$\varphi_1 \vee \varphi_2$	$[\![\varphi_1]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} \cup [\![\varphi_2]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}$
$\varphi_1 \wedge \varphi_2$	$[\![\varphi_1]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} \cap [\![\varphi_2]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}$
$\exists p.\varphi$	$\{u \mid \text{for some } (u,p,v) \in \mathcal{G}, v \in [\![\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}\}$
$\forall p.\varphi$	$\{u \mid \text{for all } (u,p,v) \in \mathcal{G}, v \in [\![\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}\}$

Table 1: Semantics of shapes.

2.2 A Simple Shape Language

Consider a countable set Names of *shape names*. A *shape* φ is defined by the following syntax, where $s \in \mathsf{Names}$ is a shape name, $p \in \mathsf{IRIs}$ is a predicate, and $c \in \mathsf{IRIs} \cup \mathsf{Literals}$ is a constant.

$$\varphi ::= \bot \mid \top \mid \mathsf{test}(c) \mid s \mid \neg\varphi \mid \varphi \vee \varphi \mid \varphi \wedge \varphi \mid \exists p.\varphi \mid \forall p.\varphi.$$

A *simple shape language catalogue* or *SSL catalogue* $\mathcal{C}$ is a partial function mapping shape names to shapes such that $\mathsf{dom}(\mathcal{C})$ is finite. We say that the shape name s is *declared* in $\mathcal{C}$ if $\mathcal{C}(s)$ is defined; that is, $s \in \mathsf{dom}(\mathcal{C})$. We require that every shape name that appears in a shape in (the range of) $\mathcal{C}$ is also declared in $\mathcal{C}$. For convenience, we usually view $\mathcal{C}$ as the set of *shape declarations* of the form $s : \varphi$ such that $\mathcal{C}(s) = \varphi$. By $\mathsf{Const}(\mathcal{C})$ we denote the set of constants c such that $\mathsf{test}(c)$ appears in some shape in $\mathcal{C}$.

The semantics of a catalogue $\mathcal{C}$ on a graph $\mathcal{G}$ is defined in terms of *shape assignments*, which indicate which shape names hold in which nodes of $\mathcal{G}$. A *shape assignment* for $\mathcal{C}$ and $\mathcal{G}$ is a relation $\alpha \subseteq \mathsf{dom}(\mathcal{C}) \times (\mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C}))$. We define the semantics of shape φ in catalogue $\mathcal{C}$ on graph $\mathcal{G}$ under assignment α inductively on the structure of φ as the set $[\![\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} \subseteq \mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C})$ in Table 1. We write $\alpha(s)$ for the set $\{u \mid (s,u) \in \alpha\}$. We call α *correct* if $\alpha(s) = [\![\mathcal{C}(s)]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}$ for every $s \in \mathsf{dom}(\mathcal{C})$.

Example 1. *Consider the graph $\mathcal{G}$ and shape catalogue $\mathcal{C}$ in Fig. 1. Intuitively, the shape $\mathcal{C}(s_1)$ captures the literal*

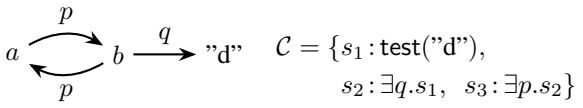

Figure 1: A graph $\mathcal{G}$ and a catalogue $\mathcal{C}$ with $a, b, p, q \in \mathsf{IRIs}$, "d" $\in \mathsf{Literals}$, and $s_1, s_2, s_3 \in \mathsf{Names}$.

"d", $\mathcal{C}(s_2)$ nodes having a q-edge to a node of shape s_1, and $\mathcal{C}(s_3)$ nodes that have a p-edge to a node of shape s_2. The assignment $\alpha_1 = \{(s_1, \text{"d"}), (s_2, b), (s_3, a)\}$ is correct. The assignment $\alpha_2 = \{(s_1, \text{"d"}), (s_2, b), (s_3, a), (s_3, b)\}$ is not correct because of the pair (s_3, b). Indeed, under assignment α_2, only the node a has a p-edge to a node assigned to s_2. That is, $\alpha_2(s_3) = \{a, b\} \neq \{a\} = [\![\mathcal{C}(s_3)]\!]^{\alpha_2}_{\mathcal{G},\mathcal{C}}$.

Next, we consider an example of a recursive catalogue.

Example 2. *Consider graph $\mathcal{G}$ in Fig. 1 and catalogue $\mathcal{C} = \{s : \exists p.s\}$. Then assignment $\alpha_3 = \{(s, a), (s, b)\}$ is correct.*

Indeed, the catalogue $\mathcal{C}$ from Example 2 is recursive, because the shape in the declaration of s uses s again. In general, a catalogue $\mathcal{C}$ is recursive if there is a directed cycle in the graph with nodes $\mathsf{dom}(\mathcal{C})$ and having an edge (s, t) if and only if t occurs in $\mathcal{C}(s)$.

SSL is intentionally simple, yet sufficient to illustrate how the choice of the semantics for recursion influences the expressive power of shape catalogues. As we will see later, the RDF schema languages ShEx and SHACL have more features, in particular for constraining the immediate neighbourhood of nodes (which in SSL is reduced to $\exists p.\varphi$ and $\forall p.\varphi$ for testing the existence of a predicate whose object satisfies a property). We discuss in Section 4 how our definition relates to real-world ShEx and SHACL.

2.3 Classic Recursive Semantics

We now describe three classic semantics for recursion: *supported model semantics* (SMS), *greatest fixpoint* (GFP), and *least fixpoint* (LFP). Each semantics specifies which shape assignments *conform* to the shape catalogue. The most permissive is SMS: it allows all correct shape assignments.

Definition 1. *A shape assignment α for catalogue $\mathcal{C}$ and graph $\mathcal{G}$* conforms to *$\mathcal{C}$ under the supported model semantics (*SMS-conforms*), if α is correct wrt. $\mathcal{C}$. We write $[\![\mathcal{C}]\!]^{\mathsf{SMS}}_{\mathcal{G}}$ for the set of all such α.*

The downside of the supported model semantics is that it allows multiple conforming shape assignments, which can be counter-intuitive and computationally expensive.

Example 3. *Continuing Ex. 2 with catalogue $\mathcal{C} = \{s : \exists p.s\}$, notice that in addition to α_3, also $\alpha_4 = \emptyset$ is correct.*

Arbitrarily choosing a single canonical shape assignment among the correct ones is problematic. Two natural ideas that come to mind are: be minimalistic and pick the *smallest* correct assignment (with respect to set inclusion), or be generous and pick the *largest* correct assignment. These ideas ultimately lead to two classical semantics, LFP and GFP, but getting there requires some work because in general there might be multiple smallest and multiple largest correct shape assignments, which we illustrate next.

Example 4. *Consider $\mathcal{G}$ in Fig. 1 with catalogue $\mathcal{C} = \{s : \exists p.\neg s\}$. Both assignments $\alpha_1 = \{(s, a)\}$ and $\alpha_2 = \{(s, b)\}$ are correct, but neither $\beta_1 = \emptyset$ nor $\beta_2 = \{(s, a), (s, b)\}$ are, showing that α_1 and α_2 are simultaneously smallest and largest at the same time.*

Single smallest and largest correct assignments are guaranteed in a special case: when the catalogue does not use negation.[1] This follows from the monotonicity of an associated operator "$\mapsto$", mapping shape assignments to shape assignments, defined as

$$\alpha \mapsto \left\{(s, v) \in \mathsf{dom}(\mathcal{C}) \times \mathsf{NC} \mid v \in [\![\mathcal{C}(s)]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}\right\}$$

$$\text{where } \mathsf{NC} = \mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C})\,.$$

Indeed, it is not hard to see that a shape assignment α is correct iff it is a fixpoint of this operator, that is, if $\alpha \mapsto \alpha$. When the catalogue does not use negation, the operator is monotone and, by the *Knaster–Tarski theorem* (Tarski 1955), it has unique least and greatest fixpoints. Moreover, by the *Kleene fixpoint theorem* (Tarski 1955), the least and greatest fixpoint can be computed by iterating the operator on, respectively, the empty shape assignment,

$$\emptyset \mapsto \alpha_1 \mapsto \alpha_2 \mapsto \dots\,,$$

and the "full" shape assignment $\Omega = \mathsf{dom}(\mathcal{C}) \times (\mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C}))$,

$$\Omega \mapsto \alpha_1 \mapsto \alpha_2 \mapsto \dots\,.$$

This allows us to define the least and greatest fixpoint semantics for catalogues that do not use negation.

Definition 2. *Let $\mathcal{C}$ be a catalogue that does not use negation. A shape assignment α for catalogue $\mathcal{C}$ and graph $\mathcal{G}$* conforms to *$\mathcal{C}$ under the LFP (resp. GFP) semantics, if α is the smallest (resp. largest) correct assignment w.r.t. $\mathcal{C}$. We also say that α LFP-conforms (resp. GFP-conforms) to $\mathcal{C}$.*

Examples 2 and 3 show that the LFP and GFP semantics of the same catalogue $\mathcal{C}$ can be different.

There is a canonical way of extending any semantics from catalogues without negation to catalogues where negation is used in a controlled fashion. This approach is called *stratification* and we explain it next. Consider a catalogue $\mathcal{C}$ without negation and assume that we have a unique shape assignment that conforms to $\mathcal{C}$ (as is the case for the LFP and GFP semantics). That is, we know what each shape name in $\mathcal{C}$ means. We could now allow using these shape names in declarations of new shape names, both positively and negatively, and when defining the semantics of the new shape names, treat the semantics of the old ones as fixed (called *frozen* in the literature). Let us illustrate this with an example.

Example 5. *Under LFP, the shape assignment that conforms to the catalogue $\{r : \mathsf{test}(a) \vee \exists p.r\}$ associates shape r to nodes that can reach node a using p-edges; for example, on graph $\mathcal{G}$ of Fig. 1, we get $\{(r, a), (r, b)\}$. Let us fix this meaning of r and extend the catalogue with declaration $s : \neg r \vee \exists q.s$. If we now apply the LFP semantics, with the meaning of r frozen, as described above, the conforming assignment associates s to nodes that can reach*

[1] There are also weaker sufficient conditions for monotonicity.

via q-edges a node that is safe*, in the sense that it cannot reach a via p-edges. On graph $\mathcal{G}$ in Fig. 1 we get* $\{(r,a),(r,b),(s,b),(s,\text{"d"})\}$.

Following this idea one can generalize the LFP and GFP semantics to *stratified* catalogues, defined below.

Definition 3. *A* stratification *for a catalogue $\mathcal{C}$ is a partitioning of* $\mathsf{dom}(\mathcal{C})$ *into sets $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$, called* strata*, such that for all i, declarations of shape names from stratum Σ_i only use shape names from $\Sigma_0 \cup \Sigma_1 \cup \cdots \cup \Sigma_i$, and are allowed to use negation only in expressions of the form $\neg\mathsf{test}(c)$ with $c \in \mathsf{IRIs} \cup \mathsf{Literals}$ and $\neg s$ with $s \in \Sigma_0 \cup \Sigma_1 \cup \cdots \cup \Sigma_{i-1}$. A catalogue $\mathcal{C}$ is* stratified *if there is a stratification for $\mathcal{C}$.*

In Example 5, we can take $\Sigma_0 = \{r\}$ and $\Sigma_1 = \{s\}$ for the extended catalogue. Given a stratification $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$ for a catalogue $\mathcal{C}$, we can define the conforming shape assignment inductively for shape names from Σ_i for $i = 0, 1, \ldots, n$, at each level substituting shape names from lower levels with their already computed semantics. While stratifications are not unique, the resulting LFP and GFP semantics do not depend on their choice (see, e.g., (Apt, Blair, and Walker 1988)).

Definition 4. *Consider a graph $\mathcal{G}$ and a catalogue $\mathcal{C}$ with stratification $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$. Let $\mathcal{C}_i$ be obtained by restricting $\mathcal{C}$ to $\Sigma_0 \cup \Sigma_1 \cup \cdots \cup \Sigma_i$; that is, $\mathcal{C}_i$ consists of declarations from $\mathcal{C}$ for shape names from $\Sigma_0 \cup \Sigma_1 \cup \cdots \cup \Sigma_i$. Define*

$$\alpha_i \subseteq \Sigma_i \times (\mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C}))$$

iteratively for $i = 0, 1, \ldots, n$, as follows. Assuming that $\alpha_0, \ldots, \alpha_{i-1}$ are already known, let α_i be the least shape assignment over Σ_i such that $\alpha_0 \cup \alpha_1 \cup \cdots \cup \alpha_i$ is correct for $\mathcal{C}_i$. We say an assignment α for $\mathcal{G}$ and $\mathcal{C}$ LFP-conforms *to $\mathcal{C}$ if $\alpha = \alpha_0 \cup \alpha_1 \cup \cdots \cup \alpha_n$. Notice that, by definition, α is unique. We write this α as $[\![\mathcal{C}]\!]^{\mathsf{LFP}}_{\mathcal{G}}$. The shape assignment $[\![\mathcal{C}]\!]^{\mathsf{GFP}}_{\mathcal{G}}$* GFP-conforming *to $\mathcal{C}$ is defined analogously, except that for α_i we take the largest shape assignment over Σ_i s.t. $\alpha_0 \cup \alpha_1 \cup \cdots \cup \alpha_i$ is correct for $\mathcal{C}_i$.*

Example 6. *Shape assignments that* GFP*-conform to catalogue $\{s : \neg\mathsf{test}(b) \wedge \forall p.s\}$ associate shape s to nodes that cannot reach node b using p-edges. For $\mathcal{G}$ from Fig. 1, one gets $\{(s, \text{"d"})\}$.*

The LFP and GFP semantics are tightly connected by a fundamental principle of *duality*, which allows translations back and forth. For a stratified SSL catalogue $\mathcal{C}$ we define the *dual catalogue*

$$\widetilde{\mathcal{C}} = \{s : \widetilde{\varphi} \mid (s : \varphi) \in \mathcal{C}\}$$

where the dual shape $\widetilde{\varphi}$ is defined recursively as follows, relying on φ using $\neg$ only in expressions $\neg\mathsf{test}(c)$ and $\neg s$,

$$\widetilde{\bot} = \top\,,\ \widetilde{\top} = \bot\,,\ \widetilde{\mathsf{test}(c)} = \neg\mathsf{test}(c)\,,\ \widetilde{\neg\mathsf{test}(c)} = \mathsf{test}(c)\,,$$
$$\widetilde{s} = s\,,\ \widetilde{\neg s} = \neg s\,,\ \widetilde{\varphi_1 \wedge \varphi_2} = \widetilde{\varphi_1} \vee \widetilde{\varphi_2}\,,$$
$$\widetilde{\varphi_1 \vee \varphi_2} = \widetilde{\varphi_1} \wedge \widetilde{\varphi_2}\,,\ \widetilde{\exists p.\varphi} = \forall p.\widetilde{\varphi}\,,\quad \widetilde{\forall p.\varphi} = \exists p.\widetilde{\varphi}\,.$$

That is, $\widetilde{\varphi}$ is obtained from φ by swapping $\top$ and $\bot$, $\mathsf{test}(c)$ and $\neg\mathsf{test}(c)$, $\wedge$ and $\vee$, as well as $\exists$ and $\forall$, but s and $\neg s$ are not swapped.

Example 7. *Consider again the catalogue $\{s : \neg\mathsf{test}(b) \wedge \forall p.s\}$ from Ex. 6. Shape assignments that* LFP*-conform to its dual $\{s : \mathsf{test}(b) \vee \exists p.s\}$ assign s to nodes that can reach b using p-edges.*

Proposition 1. *Let $\mathcal{G}$ be a graph and $\mathcal{C}$ a stratified* SSL *catalogue. Then, for every shape assignment α for $\mathcal{G}$ and $\mathcal{C}$,*

$$\alpha \text{ LFP-conforms to } \mathcal{C} \quad \text{iff} \quad \Omega \setminus \alpha \text{ GFP-conforms to } \widetilde{\mathcal{C}}$$

for $\Omega = \mathsf{dom}(\mathcal{C}) \times (\mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C}))$.

Proof. Suppose first that $\mathcal{C}$ has a single stratum; that is, it only uses $\neg$ in expressions of the form $\neg\mathsf{test}(c)$. Let us write $\mapsto$ for the operator associated with $\mathcal{C}$ and $\tilde{\mapsto}$ for the one associated with $\widetilde{\mathcal{C}}$. Both operators are monotone. From the definition of $\widetilde{\mathcal{C}}$ it follows that $\alpha \mapsto \alpha'$ iff $\Omega \setminus \alpha \tilde{\mapsto} \Omega \setminus \alpha'$. Hence, $\emptyset \mapsto \alpha_1 \mapsto \alpha_2 \mapsto \ldots$ iff $\Omega \mapsto \Omega \setminus \alpha_1 \tilde{\mapsto} \Omega \setminus \alpha_2 \tilde{\mapsto} \ldots$. By the Kleene Fixed-Point Theorem, α is the least fixpoint of $\mapsto$ iff $\Omega \setminus \alpha$ is the greatest fixpoint of $\tilde{\mapsto}$.

For a general stratified catalogue $\mathcal{C}$, we note that a stratification $\Sigma_0, \Sigma_1, \ldots, \Sigma_n$ for $\mathcal{C}$ is also a stratification for $\widetilde{\mathcal{C}}$, and proceed by straightforward induction following Def. 4. □

Using Proposition 1 one can show that SSL under both LFP and GFP corresponds precisely to the alternation-free fragment of the modal μ-calculus (Kozen 1983) with nominals but without atomic propositions, which means it has considerable expressive power.

2.4 Schemas, Conformance, and Validation

The real purpose of shape languages is specifying conformance of *graphs* to *shape schemas*, which ensures that graphs satisfy some specified constraints. The task of checking if a given graph conforms to a schema is called *validation*.

To this end, both SHACL and ShEx have a mechanism to select nodes that need to satisfy some shape: SHACL has *target declarations* (Knublauch and Kontokostas 2017) and ShEx has *shape maps* (Prud'hommeaux and Baker 2017). We abstract them here as *selector maps*. Given a shape catalogue $\mathcal{C}$, a selector map sel for $\mathcal{C}$ is a finite set of *selectors* of the form $s : \sigma$ with $s \in \mathsf{dom}(\mathcal{C})$ and σ is a shape that does not use shape names.[2] A *shape schema* is then a tuple $\mathcal{S} = (\mathcal{C}, sel)$ with $\mathcal{C}$ a shape catalogue, and sel a selector map. Intuitively, every $s : \sigma$ in sel expresses that every node u that has shape σ should also have shape s under shape assignment(s) conforming to $\mathcal{C}$. In order to define the semantics precisely, we need to handle the nondeterminism of SMS and agree on the domain over which u ranges.

A shape assignment for catalogue $\mathcal{C}$ and graph $\mathcal{G}$ associates shape names with elements of $\mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C})$, which does not necessarily include all constants mentioned in sel. We deal with this by incorporating the selector map into the catalogue as follows. We define $\mathcal{C}_{sel} = \mathcal{C} \cup \{t_{s,\sigma} : \sigma \wedge \neg s \mid s : \sigma \in sel\}$, where every $t_{s,\sigma}$ is a fresh shape name that depends on s and σ. Then, for a correct shape assignment α for $\mathcal{G}$ and $\mathcal{C}_{sel}$, we say a graph $\mathcal{G}$ *conforms* to schema $(\mathcal{C}, sel)$

[2] Both ShEx and SHACL impose additional restrictions on σ, essentially allowing only atomic shape expressions, and do not allow $\top$; see (Ahmetaj et al. 2025a).

under α if $\alpha(t_{s,\sigma}) = \emptyset$ for every shape name $t_{s,\sigma}$. Finally, we say that $\mathcal{G}$ LFP-*conforms* to $(\mathcal{C}, sel)$ if $\mathcal{G}$ conforms to $(\mathcal{C}, sel)$ under $\alpha = [\![\mathcal{C}_{sel}]\!]_{\mathcal{G}}^{\mathsf{LFP}}$, and analogously for GFP.

We can define graph conformance under SMS similarly, but we need to decide what to do with the multiple conforming shape assignments a catalogue might admit. Two approaches have been considered, called the *cautious* and *brave* SMS semantics (Corman, Reutter, and Savković 2018; Andresel et al. 2020; Bogaerts and Jakubowski 2021). We say that $\mathcal{G}$ *cautiously* (resp. *bravely*) SMS-*conforms to* schema $(\mathcal{C}, sel)$ if $\mathcal{G}$ conforms to $(\mathcal{C}, sel)$ under *every* $\alpha \in [\![\mathcal{C}_{sel}]\!]_{\mathcal{G}}^{\mathsf{SMS}}$ (resp. *some* $\alpha \in [\![\mathcal{C}_{sel}]\!]_{\mathcal{G}}^{\mathsf{SMS}}$). We use brave semantics by default and say simply that $\mathcal{G}$ SMS-*conforms to* $\mathcal{C}$.

Example 8. *Consider the graph* $\mathcal{G} = \{(a, p, a)\}$ *and the schema* $(\mathcal{C}, sel)$ *where* $\mathcal{C} = \{s : \exists p.s\}$ *and* $sel = \{s : \mathsf{test}(a)\}$. *Under* LFP *semantics, we would expect the shape assignment* $\alpha_l = \emptyset$, *while under* GFP *semantics, we get the shape assignment* $\alpha_g = \{(s, a)\}$. *Thus,* $\mathcal{G}$ GFP-*conforms to* $(\mathcal{C}, sel)$, *but* $\mathcal{G}$ *does not* LFP-*conform to* $(\mathcal{C}, sel)$.

3 A Comparative Study of Implementations

We present an experimental study of real-world engines for SHACL and ShEx to compare how they treat recursive schemas. The source code and data needed to reproduce the results of the study, as well as the raw results of our experiments, are available on Zenodo (Ahmetaj et al. 2025b).

We want to understand, for each engine, *is there a formal semantics for recursive shapes that explains its behaviour?* The main challenge is that real-world systems only test *whether a given graph conforms to a given schema*, without giving access to the witnessing shape assignments.

In order to detect which semantics is applied, we design test cases that aim to separate LFP, GFP, brave SMS, and cautious SMS. First, we run all test cases on all engines. If we observe inconsistent behaviour, we investigate the test results in more detail. If all answers are consistent with one of the four considered semantics, we interpret this as the answer to our question.

We designed thirteen test cases, falling into two test categories: separation and feature tests. Separation tests attempt to distinguish whether an engine uses LFP, GFP, brave SMS, or cautious SMS. Feature tests do exactly what their name suggests. Each such test $S = \langle \mathcal{G}, \mathcal{C}, sel \rangle$ consists of a graph $\mathcal{G}$, a shape catalogue $\mathcal{C}$, and a shape map sel. The intended use of S is to produce a ShEx or SHACL schema based on $\mathcal{C}$ and sel to check whether $\mathcal{G}$ conforms to (C, sel).

For each separation test we list all correct shape assignments. The unique conforming assignments under LFP and GFP appear as α_{LFP} and α_{GFP}, and the remaining ones as α_1, α_2. An assignment α is blue, if $\mathcal{G}$ conforms to $(\mathcal{C}, sel)$ under α, and red, if it does not.

Basic separation tests. These four tests are designed to distinguish a validator that employs LFP semantics from one that employs GFP, using minimal examples.

- `bsep1`:
  $\mathcal{G} = \{(a, p, a)\}$, $\mathcal{C} = \{s : \exists p.s\}$, $sel = \{s : \mathsf{test}(a)\}$,
  $\alpha_{\mathsf{LFP}} = \emptyset$, $\alpha_{\mathsf{GFP}} = \{(s, a)\}$.
- `bsep2`:
  $\mathcal{G} = \{(a, p, a), (b, p, b)\}, \mathcal{C} = \{s : \exists p.s\}$,
  $sel = \{s : \mathsf{test}(a), s : \mathsf{test}(b)\}$,
  $\alpha_{\mathsf{LFP}} = \emptyset$, $\alpha_{\mathsf{GFP}} = \{(s, a), (s, b)\}$,
  $\alpha_1 = \{(s, a)\}$, $\alpha_2 = \{(s, b)\}$.
- `bsep3`:
  $\mathcal{G} = \{(a, p, c), (b, p, c)\}, \mathcal{C} = \{s : s' \wedge \exists p,\ s' : s \wedge \exists p\}$,
  $sel = \{s : \mathsf{test}(a), s : \mathsf{test}(b)\}$,
  $\alpha_{\mathsf{LFP}} = \emptyset$, $\alpha_{\mathsf{GFP}} = \{(s, a), (s, b), (s', a), (s', b)\}$,
  $\alpha_1 = \{(s, a), (s', a)\}$, $\alpha_2 = \{(s, b), (s', b)\}$.
- `bsep4`:
  $\mathcal{G} = \{(a, p, a), (b, p, b)\}$,
  $\mathcal{C} = \{s : \exists p.s, s' : \neg s\}$, $sel = \{s : \mathsf{test}(a), s : \mathsf{test}(b)\}$,
  $\alpha_{\mathsf{LFP}} = \{(s', a), (s', b)\}$, $\alpha_{\mathsf{GFP}} = \{(s, a), (s, b)\}$,
  $\alpha_1 = \{(s, a), (s', b)\}$, $\alpha_2 = \{(s, b), (s', a)\}$.

Reachability separation tests. Then, we tested a classical recursive property of reachability and its dual property of safety by checking them on two dual shape names r and s, in four different scenarios. All scenarios use the same graph:

- `reach1`:
  $\mathcal{C} = \{r : \mathsf{test}(a) \vee \exists p.r\}$,
  $sel = \{r : \mathsf{test}(a), r : \mathsf{test}(b), r : \mathsf{test}(c), r : \mathsf{test}(d)\}$,
  $\alpha_{\mathsf{LFP}} = \{(r, a), (r, b)\}$,
  $\alpha_{\mathsf{GFP}} = \{(r, a), (r, b), (r, c), (r, d)\}$.
- `reach2`:
  $\mathcal{C} = \{s : \neg\mathsf{test}(a) \wedge \forall p.s,\ r : \neg s\}$,
  $sel = \{r : \mathsf{test}(c), r : \mathsf{test}(d)\}$,
  $\alpha_{\mathsf{LFP}} = \{(r, a), (r, b), (r, c), (r, d)\}$,
  $\alpha_{\mathsf{GFP}} = \{(r, a), (r, b), (s, c), (s, d)\}$.
- `safe1`:
  $\mathcal{C} = \{s : \neg\mathsf{test}(a) \wedge \forall p.s\}$, $sel = \{s : \mathsf{test}(c), s : \mathsf{test}(d)\}$,
  $\alpha_{\mathsf{LFP}} = \emptyset$, $\alpha_{\mathsf{GFP}} = \{(s, c)(s, d)\}$.
- `safe2`:
  $\mathcal{C} = \{r : \mathsf{test}(a) \vee \exists p.r,\ s : \neg r\}$,
  $sel = \{r : \mathsf{test}(c), r : \mathsf{test}(d)\}$,
  $\alpha_{\mathsf{LFP}} = \{(r, a), (r, b), (s, c), (s, d)\}$,
  $\alpha_{\mathsf{GFP}} = \{(r, a), (r, b), (r, c), (r, d)\}$.

Feature tests These target more specific properties of the validator, as we will explain after introducing them.

- `nstrat1` (non-stratified negation without cyclic data):
  $\mathcal{G} = \{(a, p, b), (b, p, c), (c, p, d), (d, p, e)\}, \mathcal{C} = \{s : \exists p.\neg s\}$,
  $sel = \{s : \mathsf{test}(a), s : \mathsf{test}(b), s : \mathsf{test}(c), s : \mathsf{test}(d), s : \mathsf{test}(e)\}$
  Passing Condition: Accept or reject the schema over $\mathcal{G}_{n1}$.
- `nstrat2` (non-stratified negation with cyclic data):
  $\mathcal{G} = \{(a, p, b), (b, p, a)\}, \mathcal{C} = \{s : \exists p.\neg s\}$,
  $sel = \{s : \mathsf{test}(a), s : \mathsf{test}(b)\}$
  Passing Condition: Accept or reject the schema over $\mathcal{G}$.
- `fresh` (fresh constant support):
  $\mathcal{G} = \{(a, p, b), (b, p, c), (c, p, a)\}, \mathcal{C} = \{s : \top\}$,
  $sel = \{s : \mathsf{test}(d)\}$
  Passing Condition: The validator accepts.
- `cons1` (consistency): $\mathcal{G} = \{(a, p, a)\}, \mathcal{C} = \{s : \exists p.s', s' : \neg s\}$,
  $sel = \{s : \mathsf{test}(a), s' : \mathsf{test}(a)\}$
  Passing Condition: The validator rejects.

Test instance	rudof (ShEx)	Jena ShEx	ShEx-S	pySHACL	SHACL-S	Jena SHACL	Topbraid	rudof (SHACL)	GFP	LFP	bSMS	cSMS
bsep1	●	●	●	●	●	●	●	●	●	■	●	■
bsep2	●	●	●	●	●	●	●	●	●	■	●	■
bsep3	●	●	●	X	X	●	■	X	●	■	●	■
bsep4	●	●	●	●	●	●	●	●	●	■	●	■
reach1	●	●	●	●	●	●	■	●	●	■	●	■
reach2	■	■	■	●	●	●	●	X	■	●	●	■
safe1	●	●	●	●	●	●	●	X	●	■	●	■
safe2	●	●	●	●	●	●	■	●	●	■	●	■
nstrat1	✗*	✓	✗*	✓	✓	✓	✓	✓	-	-	-	-
nstrat2	✗*	✓	✗*	✓	✓	✓	✓	✓	-	-	-	-
fresh	✗	✓	✓	✓	✓	✓	✓	✓	-	-	-	-
cons1	✗*	✓	✗*	✓	✓	✓	✓	✓	-	-	-	-
cons2	✗*	✗	✗*	✓	✓	✓	✓	✓	-	-	-	-

Table 2: Results of comparative study of the semantics of SHACL and ShEx validators for recursive shape catalogues.

- `cons2` (consistency): $\mathcal{G}$ as in `nstrat2`, $\mathcal{C}$ as in `cons1`, $sel = \{s : \mathsf{test}(a), s' : \mathsf{test}(a), s : \mathsf{test}(b), s' : \mathsf{test}(b)\}$
  Passing Condition: The validator rejects.

The tests `nstrat1` and `nstrat2` are for shape catalogues that do not permit a stratification as in Definition 3. While none of our semantics support this setting, we still aim to find out how real-world validators behave on such inputs. The test `fresh` has a shape catalogue with a shape assignment that is trivially satisfied, using the shape $\top$. In the shape map, we require that this shape is satisfied in a "fresh node", that is, a node that is not featured in the graph. This reflects something that both SHACL and ShEx permit: selecting nodes outside the input graph. Finally, `cons1` and `cons2` check for *logical consistency*. We have a catalogue with two shapes, s and s', where s' is defined as the negation of s. Then we ask in the shape map that both s and s' be assigned to the same node. Since no such assignment is possible while being consistent with the catalogue's semantics, this test requires that a validator rejects.

Validation Engines Since the SHACL W3C Recommendation does not require SHACL validators to support recursive schemas, many validators reject them all or raise a warning indicating that recursion is not supported. We focus on those that do support recursive SHACL schemas or where we can ignore the warnings and perform the validation anyway, meaning that they accept a non-empty set of graphs when validating against a recursive SHACL schema. These are pySHACL (pySHACL 2025), SHACL-S (SHACL-S 2024), Jena SHACL (Apache Jena 2025), Topbraid (TopQuadrant SHACL 2025), and rudof (RuDoF Project 2025), which supports both SHACL and ShEx. For ShEx, we have ShEx-S (ShEx-S 2025), Jena ShEx (Apache Jena 2025), and rudof (RuDoF Project 2025).

Discussion We see the results of our experiments in Table 2. For each of the separation tests (first eight rows), a validator can answer *yes* (●) or *no* (■), or fail (X) by reporting an error, crashing, or not terminating within 5 seconds.[3] In the columns GFP, LFP, bSMS (brave SMS), and cSMS (cautious SMS), we indicate the answer prescribed by the respective semantics. For `bsep1`, for example, $\mathcal{G}$ does not conform to $(\mathcal{C}, sel)$ under α_{LFP}, so the correct answer under LFP is no (■). However, $\mathcal{G}$ does conform to $(\mathcal{C}, sel)$ under α_{GFP}, so the correct answer under GFP is yes (●). The answer yes is also correct under bSMS, because α_{GFP} witnesses it. However, the answer no is correct under cSMS because α_{LFP} witnesses it: it is a correct assignment that does not associate s to a as required by sel. An analogous reasoning applies for all values of yes (●) and no (■) in the last four columns.

Therefore, using Table 2, we can immediately see which validators are consistent with which semantics. We see that the three ShEx validators, rudof, Jena ShEx and ShEx-S, are consistent with GFP, as prescribed by the official semantics (Prud'hommeaux et al. 2019). On the SHACL side, the picture is more varied. Four validators, namely pySHACL, SHACL-S, Jena SHACL and rudof, are consistent with bSMS, if we ignore the error cases. The behaviour of Topbraid stands out. It is not consistent with either GFP, LFP, bSMS, or cSMS, as can be seen. Topbraid thus seems to follow a unique semantics.

For the five feature tests we have two outcomes, pass (✓) or fail (✗), since by design these tests interpret errors unrelated to validation as failing the test. We will nonetheless indicate failures due to these causes by an asterisk (✗*). The failures for the ShEx validators can be explained by the fact that all feature tests, except for `fresh`, use non-stratified negation, which is disallowed in ShEx. ShEx-S and rudof (ShEx) correctly reject such inputs. JenaShEx does not reject them and attempts validation. We see that it passes all tests, except for the second test for logical consistency. Perhaps this asymmetric behaviour of JenaShEx for two very similar tests can be attributed to a software bug. One notable result is that rudof does not pass `fresh`; indeed it is the only validation engine to fail it, and curiously, rudof itself, when targeting SHACL, *does* pass it. On the SHACL side, all validators pass all the feature tests.

Note that our test results do not prove that any of the validators follows a given semantics, as other tests might reveal inconsistencies. They do prove, however, that some validators do not follow a given semantics, and they give us hints as to how these real-world systems act. Claiming that their semantics follows GFP, LFP, bSMS, or cSMS requires knowledge of the algorithm each system implements, and a formal study of said algorithm.

4 Recursion in ShEx and SHACL

Building on the framework developed in Section 2, we now explore how the choice of semantics for recursion impacts the expressive power and complexity of ShEx and SHACL. We give semantics-preserving translations between core fragments of the two languages under GFP/LFP and establish

[3] Our test graphs have at most 5 nodes and edges, so the timeout is rather generous.

tight complexity bounds exposing the cost of SMS.

4.1 The ShEx and SHACL Schema Languages

We instantiate our abstract framework with the standard RDF validation languages ShEx (Prud'hommeaux et al. 2019) and SHACL (Knublauch and Kontokostas 2017). We see them as variants of SSL that allow different kinds of shapes: most notably, SHACL shapes use complex path expressions to specify paths in terms of sequences of predicates, and ShEx shapes use triple expressions to specify neighbourhoods in terms of sets of triples. The notions of shape catalogues and shape assignments naturally extend to ShEx and SHACL. Below we present the syntaxes of these languages as they were presented in (Ahmetaj et al. 2025a), to which we add recursion via references to shape names, as in SSL. Their semantics are combinations of the semantics of (non-recursive) shapes from (Ahmetaj et al. 2025a) with the semantics of recursion we have defined for SSL.

Both languages use *value type* constraints that are interpreted as sets of constants. Formally, $\mathcal{T}$ is a finite set of value types, and for every $\tau \in \mathcal{T}$, its semantics is $[\![\tau]\!] \subseteq \mathsf{IRIs} \cup \mathsf{Literals} \cup \mathsf{Blanks}$. Throughout this subsection, we fix an RDF graph $\mathcal{G}$.

Definition 5 (SHACL syntax and semantics)**.** *SHACL* shapes φ *and* path expressions π *are given by the following grammar, with* $c \in \mathsf{IRIs} \cup \mathsf{Literals}$, $\tau \in \mathcal{T}$, $Q \subseteq_{\mathit{fin}} \mathsf{IRIs}$, $p \in \mathsf{IRIs}$, $n \in \mathbb{N}$, $s \in \mathsf{Names}$*:*

$$\varphi ::= \top \mid \bot \mid \mathsf{test}(c) \mid \mathsf{test}(\tau) \mid s \mid \mathsf{closed}(Q) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \mathsf{eq}(\pi, p) \mid \mathsf{disj}(\pi, p) \mid \exists^{\geq n}\pi.\varphi \mid \exists^{\leq n}\pi.\varphi \,.$$

$$\pi ::= \mathsf{id} \mid p \mid \pi^- \mid \pi \cdot \pi \mid \pi \cup \pi \mid \pi^* \,.$$

The semantics $[\![\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} \subseteq \mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C})$ *of a SHACL shape* φ *in catalogue* $\mathcal{C}$ *on graph* $\mathcal{G}$ *under assignment* α *is defined by recursion on the structure of* φ*, by extending Table 1 with*

$$[\![\mathsf{test}(\tau)]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} = [\![\tau]\!],$$
$$[\![\exists^{\geq n}\pi.\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} = \{v \mid \#\{u \in [\![\pi]\!]^{\mathcal{G}}_v \mid u \in [\![\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}\} \geq n\},$$
$$[\![\exists^{\leq n}\pi.\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} = \{v \mid \#\{u \in [\![\pi]\!]^{\mathcal{G}}_v \mid u \in [\![\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}\} \leq n\},$$
$$[\![\mathsf{closed}(Q)]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} = \{v \mid [\![p]\!]^{\mathcal{G}}_v = \emptyset \text{ for all } p \in \mathsf{IRIs} \setminus Q\},$$
$$[\![\mathsf{eq}(\pi, p)]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} = \{v \mid [\![\pi]\!]^{\mathcal{G}}_v = [\![p]\!]^{\mathcal{G}}_v\},$$
$$[\![\mathsf{disj}(\pi, p)]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} = \{v \mid [\![\pi]\!]^{\mathcal{G}}_v \cap [\![p]\!]^{\mathcal{G}}_v = \emptyset\},$$

where $[\![\pi]\!]^{\mathcal{G}}_v$ *is the set of nodes accessible in* $\mathcal{G}$ *from* v *by a sequence of edges whose predicates match the path expression* π*; that is,* $[\![\pi]\!]^{\mathcal{G}}_v = \{u \mid (v, u) \in [\![\pi]\!]^{\mathcal{G}}\}$ *and* $[\![\pi]\!]^{\mathcal{G}}$ *is as in (Ahmetaj et al. 2025a).*

Definition 6 (ShEx syntax and semantics)**.** *ShEx* shapes φ*,* triple expressions e*, and* closed triple expressions f*, are defined by the following grammar, with* $c \in \mathsf{IRIs} \cup \mathsf{Literals}$*,* $\tau \in \mathcal{T}$*,* $s \in \mathsf{Names}$*,* $p \in \mathsf{IRIs}$*, and* $R, Q \subseteq_{\mathit{fin}} \mathsf{IRIs}$*:*

$$\varphi ::= \mathsf{test}(c) \mid \mathsf{test}(\tau) \mid s \mid \{e\} \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \neg\varphi \,.$$
$$e ::= f\,;(\neg R^-)^* \mid f\,;(\neg R^-)^*\,;(\neg Q)^* \,.$$
$$f ::= \varepsilon \mid p.\varphi \mid p^-.\varphi \mid f\,;f \mid f \mid f \mid f^* \,.$$

The semantics $[\![\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} \subseteq \mathsf{Nodes}(\mathcal{G}) \cup \mathsf{Const}(\mathcal{C})$ *of a ShEx shape* φ *in catalogue* $\mathcal{C}$ *on graph* $\mathcal{G}$ *under assignment* α *is defined by recursion on the structure of* φ*, by extending Table 1 with*

$$[\![\{e\}]\!]^{\alpha}_{\mathcal{G},\mathcal{C}} = \{\, v \mid \mathsf{Neigh}^{\pm}_{\mathcal{G}}(v) \in [\![e]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}\},$$

where $\mathsf{Neigh}^{\pm}_{\mathcal{G}}(v)$ *collects edges* $(v, p, v') \in \mathcal{G}$ *outgoing from* v *and inverses* (v, p^-, v') *of edges* $(v', p, v) \in \mathcal{G}$ *incoming to* v*, and* $[\![e]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}$ *is the family of sets of edges and inverses of edges from* $\mathcal{G}$ *that match* e*, defined recursively as follows. Expression* ε *matches* $\emptyset$*. Expressions* $p.\varphi$ *and* $p^-.\varphi$ *match singletons* $\{(u, p, w)\}$ *and* $\{(u, p^-, w)\}$ *such that* $w \in [\![\varphi]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}$*. Expression* $f_1 \mid f_2$ *matches sets matched by* f_1 *and sets matched by* f_2*. Expression* $f_1\,;f_2$ *matches sets that can be expressed as a union of two disjoint sets* F_1 *and* F_2 *matched by* f_1 *and* f_2*, respectively. Expression* f^* *matches sets that can be expressed as a union of arbitrarily many disjoint sets matching* f*. Expression* $(\neg Q)^*$ *matches arbitrary sets of* p*-edges with* $p \notin Q$ *and* $(\neg R^-)^*$ *matches arbitrary sets of* p^-*-edges with* $p \notin R$*.*

Intuitively, a set of triples matched by a closed triple expression f may contain only triples explicitly mentioned in f. A closed expression cannot be used directly in a ShEx shape: it can only occur as a subexpression of an expression e which opens allowed set of triples to all p^--edges with $p \notin R$ and possibly also to all p-edges with $p \notin Q$.

Stratification for ShEx and SHACL is defined as for SSL. (Using auxiliary shape definitions, one can rewrite each schema so that it applies negation only to sub-expressions of the form $\mathsf{test}(c)$, $\mathsf{test}(\tau)$ and s.) Following standard ShEx, we assume GFP by default and say that a graph conforms to a ShEx schema if it GFP-conforms.

4.2 Expressive Power

We now identify large equi-expressive syntactic fragments of ShEx (with GFP) and SHACL with LFP that have the same expressive power. In the case of ShEx, we forbid applying '*' to triple expressions that contain ';'.

Definition 7 (Restricted ShEx)**.** Restricted ShEx *is the syntactic fragment of ShEx obtained by replacing the rule for closed triple expressions* f *from Def. 6 with the following derivation rules:*

$$f ::= f' \mid f'^* \mid f\,;f \mid f \mid f \,.$$
$$f' ::= \varepsilon \mid p.\varphi \mid p^-.\varphi \mid f' \mid f' \,.$$

In the case of SHACL, we restrict counting to single edges, but allow existential quantification over arbitrary path expressions. We also disallow eq and disj, as they are not expressible in ShEx: they can be used to enforce or forbid a diamond pattern, whereas ShEx shapes are invariant under unravelling the graph.

Definition 8 (Restricted SHACL)**.** Restricted SHACL *is the syntactic fragment of SHACL defined by the grammar*

$$\varphi ::= \top \mid \bot \mid \mathsf{test}(c) \mid \mathsf{test}(\tau) \mid s \mid \mathsf{closed}(Q) \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists^{\geq n}p.\varphi \mid \exists^{\geq n}p^-.\varphi \mid \exists^{\leq n}p.\varphi \mid \exists^{\leq n}p^-.\varphi \mid \exists\pi.\varphi \,.$$

	Data complexity		Combined complexity	
	LFP/GFP	SMS	LFP/GFP	SMS
SSL	P †	NP ‡	P †	NP ‡
SHACL	P †	NP ‡	P †	NP ‡
ShEx	P (Th. 2)	NP (Th. 2)	P^{NP} (Th. 3)	NP^{NP} (Th. 3)

†(Andresel et al. 2020) ‡(Corman, Reutter, and Savković 2018)

Table 3: Complexity of validation

Having defined the restricted fragments, we can formulate the equivalence result.

Theorem 1. *Restricted ShEx (with* GFP*) and restricted SHACL with* LFP *have the same expressive power and there are effective back-and-forth translations between them.*

The proof proceeds by normalizing both formalisms to even simpler fragments and applying the duality principle (see (Ahmetaj et al. 2026)).

4.3 Complexity

An overview of prior and new results on the complexity of validation is given in Table 3. Both the combined and data complexity have been known for SHACL under LFP and SMS (Andresel et al. 2020; Corman, Reutter, and Savković 2018). For GFP the complexity is the same, as it comes from iterating a polynomial number of times the same operator as for LFP. The upper bounds cover SSL, which is a fragment of SHACL. The NP lower bounds carry over too, because the SHACL schemas used in the proofs are readily expressible in SSL, but we found a much simpler proof (for a fixed schema), by a straightforward reduction from 3COLOURABILITY (see (Ahmetaj et al. 2026)). For ShEx it has only been known that combined complexity for the fragment without Boolean connectives is NP-complete under GFP (Staworko et al. 2015). We fill this gap by establishing tight bounds for full ShEx under all three semantics.

Theorem 2 (Data complexity of ShEx)**.** *For a fixed ShEx schema $\mathcal{S}$, the problem of deciding whether a given graph $\mathcal{G}$ conforms to $\mathcal{S}$ is in* P *for both* LFP *and* GFP*, and* NP*-complete for* SMS*.*

The NP lower bound for SMS in Theorem 2 carries over from SSL which is subsumed by ShEx. Let us move to the upper bounds.

The validation problem for all three semantics amounts to applying the operator over shape assignments that is associated with the shape catalogue: for SMS we apply the operator to the guessed shape assignment to check if it is a fixpoint, and for LFP and GFP we iterate the operator over a suitable initial shape assignment until a fixpoint is reached, which is guaranteed to happen in a polynomial number of iterations. Applying the operator to a shape assignment α amounts to evaluating shape expressions in the catalogue, which is straightforward (following their syntactic structure) except for one subtask: checking whether the neighbourhood of a node in the graph matches a triple expression. Thus, in order to prove the upper bounds from Theorem 2, it suffices to show that this subtask is tractable. We do it in Proposition 2. By adding auxiliary declarations to the catalogue, we can assume that triple expressions are *shallow*, in the sense that the shape expression φ in atomic triple expressions $p.\varphi$ and $p^-.\varphi$ must be a shape name.

Algorithm 1: Matching Triple Expressions

Input: graph $\mathcal{G}$, shape assignment α, node v, shallow triple expression $e = f_0\,;(\neg R^-)^*$ or $e = f_0\,;(\neg R^-)^*\,;(\neg Q)^*$

Output: *true* iff $\mathsf{Neigh}^{\pm}_{\mathcal{G}}(v) \in [\![e]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}$

```
M_0 := M^{α,e}_{G,v} ;     // extract multiset from Neigh^±_G(v)
return RecMatch(M_0, e)
Function RecMatch(M, f):
  if f = f_1 ; f_2 then
    Nondeterministically split M into M_1 and M_2 ;
    return RecMatch(M_1, f_1) && RecMatch(M_2, f_2)
  if f = u* then
    Nondeterministically split M into M_1 and M_2 ;
    return RecMatch(M_1, u) && RecMatch(M_2, f)
  if f = f_1 | f_2 then
    return RecMatch(M, f_1) || RecMatch(M, f_2)
  if f = q.s and M is the singleton of (q, T) with s ∈ T
    then return true ;
  if f = ¬Q and M is the singleton of (p, T) with p ∉ Q
    then return true ;
  if f = ¬R⁻ and M is the singleton of (p⁻, T) with
    p ∉ R then return true ;
  if f = ε and M is empty then return true ;
  return false
```

Proposition 2. *Fix a catalogue $\mathcal{C}$ and a shallow triple expression e as in Def. 6 occurring in $\mathcal{C}$. Given as input a graph $\mathcal{G}$, a shape assignment α for $\mathcal{G}$ and $\mathcal{C}$, and a node v, we can decide in polynomial time whether* $\mathsf{Neigh}^{\pm}_{\mathcal{G}}(v) \in [\![e]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}$.

Proof. In this proof, q stands for a predicate p or an inverse predicate p^-. Let Π_e be the set of predicates p used in e (including those in subexpressions $\neg R^-$ and $\neg Q$) and their inverses p^-, and let other be a fresh predicate not used in e. Let $\Gamma_{e,q}$ be the set of shape names s s.t. $q.s$ occurs in e.

An *edge type* for e is a pair (q, T) where $q \in \Pi_e \cup \{\mathsf{other}, \mathsf{other}^-\}$ and $T \subseteq \Gamma_{e,q}$. Let Σ_e denote the set of all edge types for e. In particular, $(\mathsf{other}, \emptyset), (\mathsf{other}^-, \emptyset) \in \Sigma_e$. Note that Σ_e might be exponential in the size of e, but this is fine as we consider e to be fixed.

A triple (v, q, v') in $\mathsf{Neigh}^{\pm}_{\mathcal{G}}(v)$ with $q \in \Pi_e$ has type (q, T) under shape assignment α if $v' \in \alpha(s)$ for each $s \in T$ and $v' \notin \alpha(s)$ for each $s \in \Gamma_{e,q} \setminus T$. A triple (v, q, v') in $\mathsf{Neigh}^{\pm}_{\mathcal{G}}(v)$ with $q \notin \Pi_e$ has type $(\mathsf{other}, \emptyset)$ if q is a predicate, and $(\mathsf{other}^-, \emptyset)$ if q is an inverse predicate. Note that each triple in $\mathsf{Neigh}^{\pm}_{\mathcal{G}}(v)$ has exactly one type.

We define $\mathcal{M}^{\alpha,e}_{\mathcal{G},v}$ as the multiset of types of triples in $\mathsf{Neigh}^{\pm}_{\mathcal{G}}(v)$. That is, $\mathcal{M}^{\alpha,e}_{\mathcal{G},v}$ is a multiset over Σ_e and the multiplicity of (q, T) in $\mathcal{M}^{\alpha,e}_{\mathcal{G},v}$ is equal to the number of triples (v, q, v') in $\mathsf{Neigh}^{\pm}_{\mathcal{G}}(v)$ that have type (q, T).

Let M be a multiset over Σ_e. For $(q, T) \in \Sigma_e$ we write $M(q, T)$ for the multiplicity of (q, T) in M. We call M *the*

singleton of (q, T) if $M(q, T) = 1$ and $M(q', T') = 0$ for all $(q', T') \in \Sigma_e \setminus \{(q, T)\}$. We say M is *split* into multisets M_1 and M_2 over Σ_e if $M_1(q, T) + M_2(q, T) = M(q, T)$ for every $(q, T) \in \Sigma_e$.

Consider the nondeterministic Algorithm 1, which recursively splits $\mathcal{M}^{\alpha,e}_{\mathcal{G},v}$ while attempting to match subexpressions of e. It is easy to see that $\mathsf{Neigh}^{\pm}_{\mathcal{G}}(v) \in [\![e]\!]^{\alpha}_{\mathcal{G},\mathcal{C}}$ iff

some run of Algorithm 1 on $\mathcal{G}, \alpha, v, e$ returns *true*. (†)

Due to the possible presence of $*$ in e, Algorithm 1 need not terminate. Yet, condition (†) can be effectively checked. In fact, it can be checked in polynomial time when e is assumed to be fixed and only the size of $\mathcal{G}, \alpha, v$ may grow, because the recursive subroutine RecMatch can be seen as an *alternating* procedure that requires only logarithmic space. Indeed, storing the multisets M, M_1, M_2 used in the RecMatch requires only logarithmic space: for each multiset we only need $|\Sigma_e|$ counters that count up to the number of nodes in $\mathcal{G}$. Hence, the problem of checking (†) is in ALOGSPACE and hence in P since ALOGSPACE = P. □

For the combined complexity the upper bounds are straightforward, and the lower bounds use reductions from suitable variants of the SAT problem (see (Ahmetaj et al. 2026)).

Theorem 3 (Combined complexity of ShEx)**.** *The problem of deciding whether a given graph $\mathcal{G}$ conforms to a given ShEx schema is* P^{NP}*-complete for* LFP *and* GFP*, and* $\mathrm{NP}^{\mathrm{NP}}$*-complete for* SMS*.*

5 Conclusion

The behaviour of SHACL validators on recursive schemas is unpredictable, which is only natural since its language design does not settle how to deal with recursion. Indeed, while ShEx systems consistently behave in line with GFP, SHACL validators mostly appear to be aligned with brave SMS, but some fail on simple schemas and others appear to mix behaviours depending on structural properties of the schema. This heterogeneity explains long-standing interoperability glitches and motivates standardizing an explicit, implementation-independent recursion contract.

Our results show that LFP, the hot candidate for recursion in SHACL, is a sensible choice:

1. It is natural. It is the semantics of common rule languages such as Datalog, the language Rel that is used in industry applications (Aref et al. 2025), as well as RDF-specific ones like SPIN (Knublauch, Hendler, and Idehen 2011) and RIF (Kifer and Boley 2013). It is also the standard semantics for stratified programs in logic programming.
2. It allows effective translations between large fragments of ShEx and SHACL, which improves interoperability and compatibility between the two languages.
3. It avoids the NP-hardness of validation under SMS.

Acknowledgements

This work was initiated during Dagstuhl Seminar 24102: *Shapes in Graph Data: Theory and Implementation*. It was funded by SHAKIGRA: Shaping Knowledge and Interoperable Graphs, code: PID2024-157010OB-I00 from the Spanish Research Agency, project SEK-25-GRU-GIC-24-089 from Asturias Regional research Plan and COST Action CA23147 GOBLIN (Labra Gayo); Austrian Science Fund (FWF) and netidee SCIENCE [T1349-N], and the Vienna Science and Technology Fund (WWTF) [10.47379/ICT2201] (Ahmetaj); Poland's NCN grant 2018/30/E/ST6/00042 (Murlak), the Austrian Science Fund (FWF) projects 10.55776/COE12 and P30873 (Šimkus); and partially supported by the Province of Bolzano and FWF through project OnTeGra (DOI 10.55776/PIN8884924, Savković).

AI Declaration

The authors have not employed any Generative AI tools.

References

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

Ahmetaj, S.; David, R.; Ortiz, M.; Polleres, A.; Shehu, B.; and Šimkus, M. 2021. Reasoning about explanations for non-validation in SHACL. In *KR 2021*, 12–21.

Ahmetaj, S.; David, R.; Polleres, A.; and Šimkus, M. 2022a. Repairing SHACL constraint violations using answer set programming. In *ISWC 2022*, 375–391. Springer.

Ahmetaj, S.; Löhnert, B.; Ortiz, M.; and Šimkus, M. 2022b. Magic shapes for SHACL validation. *Proc. VLDB Endow.* 15(10):2284–2296.

Ahmetaj, S.; Boneva, I.; Hidders, J.; Hose, K.; Jakubowski, M.; Labra Gayo, J. E.; Martens, W.; Mogavero, F.; Murlak, F.; Okulmus, C.; Polleres, A.; Savković, O.; Šimkus, M.; and Tomaszuk, D. 2025a. Common foundations for SHACL, ShEx, and PG-Schema. In *WWW 2025*, 8–21. ACM.

Ahmetaj, S.; Boneva, I.; Hidders, J.; Jakubowski, M.; Labra Gayo, J. E.; Martens, W.; Mogavero, F.; Murlak, F.; Okulmus, C.; Savković, O.; Šimkus, M.; and Tomaszuk, D. 2025b. Experimental test suite for the paper "Common Foundations for Recursive Shape Languages". https://doi.org/10.5281/zenodo.17249934.

Ahmetaj, S.; Boneva, I.; Hidders, J.; Jakubowski, M.; Labra Gayo, J. E.; Martens, W.; Mogavero, F.; Murlak, F.; Okulmus, C.; Savković, O.; Šimkus, M.; and Tomaszuk, D. 2026. Common foundations for recursive shape languages. *CoRR* abs/2604.20946.

Andresel, M.; Corman, J.; Ortiz, M.; Reutter, J. L.; Savković, O.; and Šimkus, M. 2020. Stable model semantics for recursive SHACL. In *WWW 2020*, 1570–1580. ACM / IW3C2.

Angles, R.; Bonifati, A.; Dumbrava, S.; Fletcher, G.; Hare, K. W.; Hidders, J.; Lee, V. E.; Li, B.; Libkin, L.; Martens, W.; Murlak, F.; Perryman, J.; Savković, O.; Schmidt, M.; Sequeda, J. F.; Staworko, S.; and Tomaszuk, D. 2021. PG-Keys: Keys for property graphs. In *SIGMOD 2021*, 2423–2436. ACM.

Angles, R.; Bonifati, A.; Dumbrava, S.; Fletcher, G.; Green, A.; Hidders, J.; Li, B.; Libkin, L.; Marsault, V.; Martens,

W.; Murlak, F.; Plantikow, S.; Savković, O.; Schmidt, M.; Sequeda, J. F.; Staworko, S.; Tomaszuk, D.; Voigt, H.; Vrgoč, D.; Wu, M.; and Zivkovic, D. 2023. PG-Schema: Schemas for property graphs. *Proc. ACM Manag. Data* 1(2):198:1–198:25.

Angles, R.; Thakkar, H.; and Tomaszuk, D. 2020. Mapping RDF databases to property graph databases. *IEEE Access* 8:86091–86110.

Anuradha; Swathi; Nagpal, A.; Chaturvedi, P.; Kalra, R.; and Alwan, A. A. 2023. Deep learning for anomaly detection in large-scale industrial data. In *UPCON 2023*, volume 10, 1551–1556.

Apache Jena. 2025. Apache Jena. https://jena.apache.org/. Accessed: 2025-10-04.

Apt, K. R.; Blair, H. A.; and Walker, A. 1988. Towards a theory of declarative knowledge. In *Foundations of Deductive Databases and Logic Programming 1988*. Morgan Kaufmann. 89–148.

Aref, M.; Guagliardo, P.; Kastrinis, G.; Libkin, L.; Marsault, V.; Martens, W.; McGrath, M.; Murlak, F.; Nystrom, N.; Peterfreund, L.; Rogers, A.; Sirangelo, C.; Vrgoč, D.; Zhao, D.; and Zreika, A. 2025. Rel: A programming language for relational data. In *SIGMOD Conference Companion 2025*, 283–296. ACM.

Baader, F. 1990. Terminological cycles in KL-ONE-based knowledge representation languages. In *AAAI 1990*, 621–626. AAAI Press / The MIT Press.

Baader, F. 1996. Using automata theory for characterizing the semantics of terminological cycles. *Ann. Math. Artif. Intell.* 18(2-4):175–219.

Bogaerts, B., and Jakubowski, M. 2021. Fixpoint semantics for recursive SHACL. In *ICLP Technical Communications*, EPTCS 2021, 41–47.

Bogaerts, B.; Jakubowski, M.; and Van den Bussche, J. 2022. SHACL: A description logic in disguise. In *LPNMR 2022*, 75–88. Springer.

Bogaerts, B.; Jakubowski, M.; and Van den Bussche, J. 2024. Expressiveness of SHACL features and extensions for full equality and disjointness tests. *Log. Methods Comput. Sci.* 20(1).

Boneva, I.; Labra Gayo, J. E.; and Prud'hommeaux, E. G. 2017. Semantics and validation of shapes schemas for RDF. In *ISWC 2017 (1)*, 104–120. Springer.

Bray, T.; Paoli, J.; Sperberg-McQueen, C. M.; Maler, E.; and Yergeau, F. 2008. Extensible markup language (XML) 1.0 (fifth edition). W3C Recommendation, W3C. https://www.w3.org/TR/xml/.

Corman, J.; Florenzano, F.; Reutter, J. L.; and Savković, O. 2019. Validating SHACL constraints over a SPARQL endpoint. In *ISWC 2019 (1)*, 145–163. Springer.

Corman, J.; Reutter, J. L.; and Savković, O. 2018. Semantics and validation of recursive SHACL. In *ISWC 2018 (1)*, 318–336. Springer.

De Giacomo, G., and Lenzerini, M. 1994. Concept language with number restrictions and fixpoints, and its relationship with mu-calculus. In *ECAI 1994*, 411–415. John Wiley and Sons, Chichester.

Delva, T.; Dimou, A.; Jakubowski, M.; and Van den Bussche, J. 2023. Data provenance for SHACL. In *EDBT 2023*, 285–297. OpenProceedings.org.

Deutsch, A.; Francis, N.; Green, A.; Hare, K. W.; Li, B.; Libkin, L.; Lindaaker, T.; Marsault, V.; Martens, W.; Michels, J.; Murlak, F.; Plantikow, S.; Selmer, P.; van Rest, O.; Voigt, H.; Vrgoč, D.; Wu, M.; and Zemke, F. 2022. Graph pattern matching in GQL and SQL/PGQ. In *SIGMOD 2022*, 2246–2258. ACM.

Ferranti, N.; Francisco de Souza, J.; Ahmetaj, S.; and Polleres, A. 2024. Formalizing and validating Wikidata's property constraints using SHACL and SPARQL. *Semantic Web* 15(6):2333–2380.

Gao, S. S.; Sperberg-McQueen, C. M.; Thompson, H. S.; Mendelsohn, N.; Beech, D.; and Maloney, M. 2012. W3C XML schema definition language (XSD) 1.1 part 1: Structures. W3C Recommendation, W3C. https://www.w3.org/TR/xmlschema11-1/.

Hartig, O. 2014. Reconciliation of RDF* and property graphs. *CoRR* abs/1409.3288.

ISO/IEC 39075. 2024. ISO/IEC 39075:2024 Information technology – Database languages – GQL. Standard, International Organization for Standardization.

Kifer, M., and Boley, H. 2013. RIF overview (second edition). Technical report, World Wide Web Consortium.

Knublauch, H., and Kontokostas, D. 2017. Shapes constraint language (SHACL). W3C Recommendation, W3C. https://www.w3.org/TR/shacl/.

Knublauch, H.; Hendler, J. A.; and Idehen, K. 2011. SPIN — overview and motivation. Technical report, World Wide Web Consortium.

Kozen, D. 1983. Results on the propositional mu-calculus. *Theor. Comput. Sci.* 27:333–354.

Labra Gayo, J. E.; Prud'hommeaux, E. G.; Boneva, I.; and Kontokostas, D. 2017. *Validating RDF Data*. Synthesis Lectures on the Semantic Web: Theory and Technology. Morgan & Claypool Publishers.

Labra Gayo, J. E.; García-González, H.; Fernández-Alvarez, D.; and Prud'hommeaux, E. G. 2019. *Challenges in RDF Validation*. Springer International Publishing. 121–151.

Labra Gayo, J. E.; Knublauch, H.; and Kontokostas, D. 2024. SHACL test suite and implementation report. W3C Document. https://w3c.github.io/data-shapes/data-shapes-test-suite/.

Labra Gayo, J. E. 2022. WShEx: A language to describe and validate wikibase entities. In *Wikidata@ISWC 2022*, CEUR Workshop Proceedings. CEUR-WS.org.

Labra Gayo, J. E. 2024. Extending shape expressions for different types of knowledge graphs. In *DQMLKG 2024*, CEUR Workshop Proceedings. CEUR-WS.org.

Lassila, O.; Schmidt, M.; Hartig, O.; Bebee, B.; Bechberger, D.; Broekema, W.; Khandelwal, A.; Lawrence, K.; López Enríquez, C. M.; Sharda, R.; and Thompson, B. B.

2023. The onegraph vision: Challenges of breaking the graph model lock-in. *Semantic Web* 14(1):125–134.

Leinberger, M.; Seifer, P.; Rienstra, T.; Lämmel, R.; and Staab, S. 2020. Deciding SHACL shape containment through description logics reasoning. In *ISWC 2020 (1)*, 366–383. Springer.

Lissandrini, M.; Rabbani, K.; and Hose, K. 2024. Mining validating shape for large knowledge graphs via dynamic reservoir sampling. In *SEBD 2024*, CEUR Workshop Proceedings, 25–34. CEUR-WS.org.

Martens, W.; Neven, F.; Niewerth, M.; and Schwentick, T. 2017. BonXai: Combining the simplicity of DTD with the expressiveness of XML schema. *ACM Trans. Database Syst.* 42(3):15:1–15:42.

Mohamed, A.; Abuoda, G.; Ghanem, A.; Kaoudi, Z.; and Aboulnaga, A. 2022. Rdfframes: knowledge graph access for machine learning tools. *VLDB J.* 31(2):321–346.

Nebel, B. 1990. Terminological reasoning is inherently intractable. *Artif. Intell.* 43(2):235–249.

Okulmus, C., and Šimkus, M. 2024. SHACL validation under the well-founded semantics. In *KR 2024*.

Oudshoorn, A.; Ortiz, M.; and Šimkus, M. 2026. Static analysis of recursive SHACL. In *KR 2026*.

Pareti, P.; Konstantinidis, G.; Mogavero, F.; and Norman, T. J. 2020. SHACL satisfiability and containment. In *ISWC 2020 (1)*, 474–493. Springer.

Pareti, P.; Konstantinidis, G.; and Mogavero, F. 2022. Satisfiability and containment of recursive SHACL. *J. Web Semant.* 74:100721.

Prud'hommeaux, E. G., and Baker, T. 2017. ShapeMap structure and language. W3C Draft Community Group Report, W3C. http://shex.io/shape-map/.

Prud'hommeaux, E. G.; Boneva, I.; Labra Gayo, J. E.; and Kellogg, G. 2019. Shape expressions language 2.1. W3C Community Group Report, W3C. http://shex.io/shex-semantics/.

Prud'hommeaux, E. G.; Labra Gayo, J. E.; and Solbrig, H. R. 2014. Shape expressions: an RDF validation and transformation language. In *SEMANTiCS 2014*, 32–40. ACM.

pySHACL. 2025. pySHACL: Python validator for SHACL. https://github.com/RDFLib/pySHACL. Accessed: 2025-10-04.

Rabbani, K.; Lissandrini, M.; and Hose, K. 2023. Extraction of validating shapes from very large knowledge graphs. *Proc. VLDB Endow.* 16(5):1023–1032.

RuDoF Project. 2025. Rudof project. https://github.com/rudof-project/rudof. Accessed: 2025-10-04.

Sakr, S.; Bonifati, A.; Voigt, H.; Iosup, A.; Ammar, K.; Angles, R.; Aref, W. G.; Arenas, M.; Besta, M.; Boncz, P. A.; Daudjee, K.; Valle, E. D.; Dumbrava, S.; Hartig, O.; Haslhofer, B.; Hegeman, T.; Hidders, J.; Hose, K.; Iamnitchi, A.; Kalavri, V.; Kapp, H.; Martens, W.; Özsu, M. T.; Peukert, E.; Plantikow, S.; Ragab, M.; Ripeanu, M.; Salihoglu, S.; Schulz, C.; Selmer, P.; Sequeda, J. F.; Shinavier, J.; Szárnyas, G.; Tommasini, R.; Tumeo, A.; Uta, A.; Varbanescu, A. L.; Wu, H.; Yakovets, N.; Yan, D.; and Yoneki, E. 2021. The future is big graphs: a community view on graph processing systems. *Commun. ACM* 64(9):62–71.

Schild, K. 1994. Terminological cycles and the propositional μ-calculus. In *KR 1994*, 509–520. Morgan Kaufmann.

SHACL-S. 2024. SHACL-S. https://github.com/weso/shacl-s.

ShEx-S. 2025. ShEx-S. https://github.com/weso/shex-s. Accessed: 2025-10-04.

Staworko, S.; Boneva, I.; Labra Gayo, J. E.; Hym, S.; Prud'hommeaux, E. G.; and Solbrig, H. R. 2015. Complexity and expressiveness of ShEx for RDF. In *ICDT 2015*, 195–211. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.

Tarski, A. 1955. A lattice-theoretical fixpoint theorem and its applications. *PJM* 5(2):285–309.

Tomaszuk, D. 2017. RDF validation: A brief survey. In *BDAS 2017*, Communications in Computer and Information Science, 344–355.

TopQuadrant SHACL. 2025. TopQuadrant SHACL. https://github.com/TopQuadrant/shacl. Accessed: 2025-10-04.

Vrgoč, D.; Rojas, C.; Angles, R.; Arenas, M.; Arroyuelo, D.; Buil-Aranda, C.; Hogan, A.; Navarro, G.; Riveros, C.; and Romero, J. 2023. MillenniumDB: An open-source graph database system. *Data Intell.* 5(3):560–610.

W3C OWL Working Group. 2012. OWL 2 web ontology language document overview (second edition). W3C Recommendation, W3C. https://www.w3.org/TR/2012/REC-owl2-overview-20121211/.