Effects of Entailment in SHACL Validation of Closed Shapes

Zenon Zacouris[1], Jin Ke[1] and Maribel Acosta[1]

[1]Technical University of Munich, Heilbronn, Germany

Abstract

The rapidly increasing volume of RDF data calls for automated validation. SHACL (SHApes Constraint Language) provides a declarative means of expressing constraints over RDF data, thereby improving data quality. Among its core features, SHACL supports a closure constraint that restricts a node to only those properties explicitly listed in the shape. However, existing approaches do not account for entailment when validating closed shapes: any property introduced by reasoning is treated as a violation, even when it follows from permitted properties. The absence of a well-defined semantics for SHACL closed shapes under entailment leads to numerous spurious violations during validation and, more broadly, prevents the combined use of SHACL validation and ontological reasoning. In this work, we investigate how to perform SHACL validation of closed shapes in the presence of reasoning while reducing such violations. We introduce ReC-SHACL, a shapes rewriting approach, which uses the ontology to extend the set of permissible properties for a SHACL closed shape. Building on this notion, we present an algorithm that entails closed shapes using inference rules applied to a minimal witness graph based on the shape's set of permissible properties. We demonstrate that entailing closed shapes in this way reduces the spurious violations introduced by reasoning, while also showing that not all violations can be remedied by performing ontology-only entailment of the shapes graph.

Keywords

SHACL, Data Quality, Closed Shapes, Ontology, RDFS Entailment, OWL Entailment, Knowledge Graphs

1. Introduction

The SHApes Constraint Language (SHACL [1]) is a declarative language for ensuring the structural integrity of data in Knowledge Graphs. As a W3C recommendation, it provides a framework for validating RDF data against a set of conditions. While RDF data is often extended by ontologies (RDFS or OWL) that define implicit knowledge, SHACL validation typically operates on the explicit triples of a data graph. This discrepancy in coverage means that validating only the explicit graph may overlook information that only becomes available after reasoning; however, performing reasoning prior to validation often introduces conflicts with one of the SHACL core constraints: closed shapes.

Closed shapes are particularly useful for rigorous data quality control under closed world assumption. A closed shape enforces a strict schema by restricting the properties of a focus node to those explicitly declared in the shape. In practice, this is essential for detecting data entry errors, schema drift, and unexpected properties arising from automated data integration. However, the "strict whitelisting" nature of closed shapes makes them highly sensitive to changes in the data graph when additional predicates are introduced by entailment regimes.

The SHACL specification currently leaves the semantics of validating closed shapes under entailment unspecified. Existing research has focused on general SHACL validation [2, 3, 4], reasoning prior to validation [5, 6], or shape rewriting based on ontologies [7, 8]. However, none of these specifically address the unique behavior of closed shapes, where the addition of logically correct triples can result in validation failures. To illustrate this, consider the following motivating example.

Motivation The conflict between closed shapes and entailment is shown in Figure 1. The closed shape in (Figure 1a) targets entities of type :Person and permits only the properties :knows and :hasDog,

Workshop on Quality of Knowledge Graphs (QKG), ESWC 2026, 11 May 2026 in Dubrovnik, Croatia.

zenon.zacouris@tum.de (Z. Zacouris); jin.ke@tum.de (J. Ke); maribel.acosta@tum.de (M. Acosta)

0009-0008-7806-2507 (Z. Zacouris); 0009-0001-8516-8894 (J. Ke); 0000-0002-1209-2868 (M. Acosta)

```
:s1 a sh:NodeShape ;
    sh:targetClass :Person ;
    sh:closed true ;
    sh:property [sh:path :knows ] ;
    sh:property [sh:path :hasDog ] .
```

(a) Closed shape

```
:Alice a :Person ; :likes :Bob .
:Bob a :Person ; :hasDog :Bully .
```

(b) RDF graph

```
:Alice a :Person ; :likes :Bob ;
    :knows :Bob .
:Bob a :Person ; :hasDog :Bully ;
    :hasPet :Bully .
```

(c) Entailed RDF graph

```
:likes rdfs:subPropertyOf :knows .
:hasDog rdfs:subPropertyOf :hasPet .
```

(d) Ontology

Figure 1: Motivating example: SHACL closed shape validation before and after RDFS entailment.

Table 1
SHACL validation results for each focus node under different reasoning settings

Shape	Focus node	Without reasoning	With reasoning	Expected result
:s1	:Alice	False	False	False
:s1	:Bob	True	False	True

as these properties are mentioned in the constraints of the shape. When validating the RDF graph in Figure 1b against this shape, the focus node :Alice triggers a violation: this node has the property :likes, which is not listed in the shape. Under RDFS semantics, the ontology in Figure 1d entails the triple (:Alice, :knows, :Bob), as shown in the entailed graph in Figure 1c. Yet, the predicate :likes remains on :Alice and is still not permitted by the closed shape, so the violation correctly persists after entailment.

However, the focus node :Bob demonstrates a more problematic case. Before entailment, :Bob conforms to the shape. After entailment, this focus node gains the property :hasPet because dog ownership implies pet ownership in the ontology (Figure 1d). Because :hasPet is not explicitly permitted in the shape, :Bob now triggers a violation. We consider this a *spurious validation result*: a node is non-conformant for possessing a property that is a direct, logical consequence of a permitted one. As shown in Table 1, current validation settings fail to align with the intuitive expected results when entailment is involved.

Problem Statement Previous work has shown that SHACL validation under entailment can produce spurious validation results [6], if the effects of entailment are not handled accordingly. Still, the problem of the validation of closed shapes under entailment has not been studied. When a SHACL shape is closed, every property of the focus node must be explicitly listed in the shape. However, entailment introduces new predicates which, while logically valid, are not allowed by the validator. Currently, there is no principled way to distinguish between a property that violates the schema and one that is simply an entailed byproduct of a valid property. In this work, we tackle the problem of SHACL validation of closed shapes under entailment to eliminate these spurious violations. We argue that if a property is logically entailed from a permitted set, it should inherit that permission.

Our Approach in a Nutshell To address these challenges, we propose a shape rewriting approach that aligns the closed-shape constraints with the logic of the underlying entailment regime. In this work, we focus on RDFS and OWL-LD [9]. Our method, ReC-SHACL (reasoning for closed SHACL shapes), automatically expands the set of permissible properties Q of a closed shape into an augmented set Q^+ by considering the underlying ontology in the data graph and the entailment regime. The core of this approach is the construction of a minimal witness graph, which serves as a skeletal representation of a node conforming to the original shape. By computing the closure of this witness graph under a specific

entailment regime, we identify all predicates that are logically valid (via entailment) from the permitted set. These inferred predicates are then added to the shape's whitelist, effectively "pre-authorizing" the triples that a reasoner will later introduce into the data graph. This transformation ensures that while the shape remains closed and rigorous, it no longer penalizes nodes for the natural logical consequences of their valid properties. Crucially, because this expansion happens at the shape level, the resulting shapes remain compliant with the SHACL standard and can be used by any existing SHACL validator.

Contributions In this work, we investigate the interaction between closed shapes and entailment regimes, presenting an algorithm to compute rewrite shapes that remain compatible with off-the-shelf validators. To the best of our knowledge, this is the first work to specifically investigate the impact of entailment on SHACL closed-shape validation and to propose a solution to resolve the resulting spurious violations. Our specific contributions are:

- **C1** An analysis of the divergence in validation results for closed shapes under entailment, distinguishing between sound and spurious violations.
- **C2** A shapes rewriting approach that extends permissible property sets based on ontological axioms and a specific entailment regime.
- **C3** An algorithm to compute these extended sets using a minimal witness graph approach, ensuring that the resulting shapes can be used with any off-the-shelf SHACL validator.
- **C4** An empirical study evaluating the effectiveness of our approach in reducing spurious violations across various entailment regimes and real-world ontologies.

2. Preliminaries

2.1. SHACL Shapes and Validation

SHACL validation requires two inputs: a *data graph* and a *SHACL schema*. We assume a triple-based model for graphs, which can be applied to RDF graphs.

Definition 1 (Data Graph). *A data graph G is a set of triples $G = \{(v, p, u)\}$. We define P_G as the set of predicates in G, i.e., $P_G = \{p \mid (v, p, u) \in G\}$. V_G is the set of nodes in G, i.e., $V_G = \{v \mid (x, p, y) \in G, x = v \vee y = v\}$.*

Definition 2 (SHACL Schema [10]). *A SHACL schema $\mathcal{S}$ is defined as a set of constraints in the form of 2-tuples $s = (sel, \varphi_s)$, where sel is a SHACL selector (Definition 3) and φ_s is a SHACL shape (Definition 4).*

The selector in a shape determines the finite set of nodes in the data graph that should be validated against a shape. Those nodes are called *focus nodes* [1].

Definition 3 (SHACL Selector [10]). *The grammar of a SHACL selector sel, for $p \in P_G$, and $c \in V_G$, is defined as follows:*

$$sel ::= \exists p.\top \mid \exists p^{-}.\top \mid is(c).$$

To express SHACL shapes, we rely on path expressions defined over predicates in the data graph. Next, we present the grammar of path expressions defined by Ahmetaj et al. [10] for graphs following a triple-based data model.

The literature [10, 11] provides slightly different definitions for the grammar of shapes. In this work, we mainly adopt the notation by Ahmetaj et al. [10], with two minor differences. First, we use $test()$ to refer to the validation of a condition (e.g., datatype, value comparison, regular expression, etc.), denoted I in Corman et al. [11]. Second, our grammar allows for references to (potentially other) constraints in the SHACL schema [1]. In the following, we define the grammar of SHACL shapes assumed in this work.

Definition 4 (SHACL Shape [10, 11]). *The grammar of a SHACL shape φ is defined as follows:*

$$\begin{aligned}\varphi ::= \top \mid is(c) \mid test(E) \mid & \quad \textit{(shapes over nodes)}\\ closed(Q) \mid eq(p,p) \mid disj(p,p) \mid & \quad \textit{(shapes with paths)}\\ \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \exists^{\geq n} p.\varphi \mid \exists^{\leq n} p.\varphi & \quad \textit{(shapes with paths and shapes)}\end{aligned}$$

where $\top$ is the trivial constraint, $c \in V_G$, E is a condition (e.g., datatype, value comparison, regular expression, etc.), $Q \subseteq_{fin} P_G$, $p \in P_G$, n is a natural number.

The validation of a node $v \in V_G$ in the data graph G against a SHACL shape φ is denoted $G, v \models \varphi$. Ahmetaj et al. [10] and Corman et al. [11] provide the semantics of SHACL shapes, which define the conditions when $G, v \models \varphi$ holds. Lastly, the process of SHACL validation is carried out by selecting the target nodes and validating them against each shape in the SHACL schema.

Definition 5 (SHACL Validation [10]). *Consider $s = (sel, \varphi)$ in a SHACL schema $\mathcal{S}$. A node $v \in V_G$ is a focus node of s, if v satisfies the sel condition in G, which is denoted as $G, v \models sel$. Furthermore, v is valid w.r.t. (or conforms to a shape) φ, if $G, v \models \varphi$. Lastly, a data graph G is valid w.r.t. $\mathcal{S}$, if for each $v \in V_G$ and $(sel, \varphi) \in \mathcal{S}$ it holds that:*

$$G, v \models sel \;\; \textit{implies} \;\; G, v \models \varphi.$$

Definition 6 (SHACL Closed Shape Validation [10]). *A node v in data graph G is valid w.r.t. the closed constraint with the set of properties Q if there is no property p of node v where $\exists o : (v, p, o) \in G$.*

$$G, v \models closed(Q) \iff \forall p \in (\mathcal{P} \backslash Q) : \neg(G, v \models \exists^{\geq 1} p.\top).$$

2.2. Entailment

To formalize the behavior of validation after entailment, we consider the entailed graph that results from applying the semantics of an entailment regime to a given data graph. In this work, we use different symbols for the data graph and the ontology to specify which parts are used later in our approach. In the following, we define the notion of an entailed graph, which encompasses all triples logically derived from a base graph and a given ontology under an entailment regime.

Definition 7 (Entailed Graph). *Let $\mathcal{R} = (A, R)$ be an entailment regime composed of a set of axiomatic triples A, and a set of entailment rules R. Given a graph G, and an ontology O, G^* is the corresponding graph entailed under the regime $\mathcal{R}$, i.e., $G \cup O \models_{\mathcal{R}} G^*$.*

In some entailment regimes, the computation of the entailed graph is operationalized through the iterative application of inference rules until no new triples can be derived. This leads to the concept of graph closure, which represents the complete set of triples obtained through a fixed-point computation.

Definition 8 (Graph Closure). *Given an entailment regime $\mathcal{R} = (A, R)$, a graph G, and an ontology O. The closure of the graph $G \cup O$ under $\mathcal{R}$-entailment, denoted $cl_{\mathcal{R}}(G \cup O)$, is obtained by applying the rules in R over $G \cup O \cup A$ until a fixpoint is reached.*

3. Impact of Entailment on Validation Results of Closed Shapes

We analyze the conditions under which the validation results for closed shapes change when moving from a base graph G to an entailed graph G^*. Specifically, a divergence occurs if the validation of a closed constraint $closed(Q)$ yields different outcomes when applied to G versus G^* following an entailment regime $\mathcal{R}$. Table 2 summarizes the impact of various entailment rules from the entailment regime OWL-LD [12], which is the focus of our work. In the following sections, we analyze the possible directions of validation divergence in the presence of entailment: loss of conformance (Section 3.1) and gain of conformance (Section 3.2) after entailment.

Table 2
Effect of OWL-LD entailment rules [9] on the validation of the closeness constraint after entailment.

Rule family	Rules	Effect	Explanation
Subproperty rules	PRP-SPO1	spurious results	Entails a new predicate for subject s based on an existing predicate of s.
Equiv. prop. rules	PRP-EQP1, PRP-EQP2	spurious results	Entails a new predicate for subject s based on an existing predicate of s.
Domain rule	PRP-DOM	spurious results	Entails a potentially new predicate (rdf:type) for subject s based on an existing predicate of s.
Range rule	PRP-RNG	sound results	Entails a new potentially predicate (rdf:type) for a node based on a predicate in which the node appears as object.
Inv. / symm. rules	PRP-SYMP, PRP-INV1, PRP-INV2	sound results	Entails a new predicate for node o based on a predicate of another node s.
Func. prop. rules	PRP-FP, PRP-IFP	sound results	Entails the predicate owl:sameAs for a node based on matching predicates across multiple nodes.
Node equality rules	EQ-REF, EQ-SYM, EQ-TRANS, EQ-REP-S, EQ-REP-P, EQ-REP-O, EQ-DIFF1	sound results	Entails a predicate for a node by propagating a predicate from another node entailed to be equal via owl:sameAs.
Transitive rules	PRP-TRP	no effect	Entails additional triples based on existing predicates but does not introduce a new predicate.
Subclass rules	CAX-SCO, CAX-EQC1, CAX-EQC2	no effect	Entails additional rdf:type triples based on existing rdf:type triples but does not introduce a new predicate.
Other rules	PRP-AP, PRP-IRP, PRP-ASYP, PRP-PDW, CLS-THING, CLS-NOTHING, CLS-NOTHING2, CAX-DW, DT-TYPE1, DT-TYPE2, DT-EQ, DT-DIFF, DT-NOT-TYPE, SCM-CLS, SCM-SCO, SCM-EQC1, SCM-EQC2, SCM-OP, SCM-DP, SCM-SPO, SCM-EQP1, SCM-EQP2, SCM-DOM1, SCM-DOM2, SCM-RNG1, SCM-RNG2	no effect	Does not introduce new predicates for a node.

3.1. Loss of Conformance after Entailment

This case captures a situation in which a focus node conforms to a closed shape in the base graph but fails after reasoning. This is because the reasoner adds triples to G to produce G^*, and a focus node may acquire new properties that were not present in the original data. Concretely, the loss of conformance occurs when the entailment of a data graph G into G^* introduces a property to a focus node that is not included in the set of permissible properties Q in a closed shape. Formally, this occurs when the following condition holds for a G, v, Q:

$$G, v \models closed(Q) \ \wedge \ G^*, v \not\models closed(Q). \tag{1}$$

We distinguish between two types of validation results: sound and spurious results.

Sound validation results A loss of conformance in a focus node v is *sound* if the predicate p in a triple $(v, p, o) \in G^* \setminus G$ is not only absent from Q, but is also not a logical consequence of any

predicate $q \in Q$ given the ontology O and the regime $\mathcal{R}$. Formally, this occurs when $\nexists q \in Q$ such that $\{(s, q, o)\} \cup O \models_{\mathcal{R}} (s, p, o)$. In other words, there is no $q \in Q$ such that the ontology implies that every instance of q is also an instance of p (e.g., q is not a sub-property of p).

In many cases, the loss of conformance accurately reflects the domain logic. For example, if an ontology defines a property as symmetric (e.g., :friendOf), and the data contains (:Alice, :friendOf, :Bob), reasoning will correctly infer (:Bob, :friendOf, :Alice). If :Bob's closed shape does not permit :friendOf, the resulting violation is a sound logical consequence of the inferred triple.

Spurious validation results We define a loss of conformance as *spurious*, if the predicate p triggering the closedness violation is either a necessary logical consequence of the existing permissions or an artifact of the reasoning regime's axioms. Formally, a triple $(v, p, o) \in G^*$ produces a spurious result if:

1. There exists a predicate $q \in Q$ such that $\{(s, q, o)\} \cup O \models_{\mathcal{R}} (s, p, o)$, meaning p is logically covered by the permission of q; or
2. The predicate p is inferred for all individuals by the regime $\mathcal{R}$ (e.g., $\mathcal{O} \models_{\mathcal{R}} (v, p, v)$ via reflexivity), such as in the case of owl:sameAs.

Typical examples of spurious validation results happen with OWL entailment, where reflexive relations (e.g., owl:sameAs) are not explicitly included in the original set of permissible predicates Q. In such cases, the SHACL validator will flag them as violations of the closedness constraint. These are *spurious* because the underlying data has not become invalid in a domain sense; rather, the logical consequences of the ontology have exceeded the static boundaries of the closed shape.

The goal of our work is to mitigate these spurious validation results.

3.2. Gain of Conformance after Entailment

This case would involve a node that violates the closedness constraint in the base graph G but somehow conforms in the entailed graph G^*. Formally, this occurs when the following condition holds for a G, v, Q:

$$G, v \not\models closed(Q) \ \wedge \ G^*, v \models closed(Q). \tag{2}$$

However, under the assumption of monotonic reasoning, this case is logically impossible. Monotonicity ensures that G^* is a superset of G ($G \subseteq G^*$); reasoning only adds triples and never removes them. For a node to "regain" conformance, the set of its properties in the data would have to shrink, or the set of permissible properties in the shape would have to expand to "cover" the existing violation. Since the SHACL shape is static and the data G^* only grows, a property that triggers a violation in G will persistently exist in G^*. Therefore, reasoning can only introduce new violations of closedness; it can never resolve existing ones. In conclusion, under monotonic reasoning (as in the studied entailment regimes in this work), this case never occurs.

4. Our Approach: ReC-SHACL

The primary objective of this work is to mitigate spurious validation results due to loss of conformance, as described in Section 3. To achieve this, we propose ReC-SHACL, an approach that generates a rewritten shapes graph $\mathcal{S}^*$ from $\mathcal{S}$ to ensure that validation of closedness remains consistent when moving from a base data graph G to an entailed graph G^*. Ideally, if a focus node v conforms to a closed shape $s \in \mathcal{S}$ in the original data G, it should continue to conform to the expanded shape $s^* \in \mathcal{S}^*$ in G^*, despite the presence of new triples produced by monotonic entailment regimes.

Figure 2 illustrates our approach. Given a SHACL shapes graph containing closed shapes, ReC-SHACL first extracts the set of permissible properties Q from each closed shape and constructs a witness graph G_Q. The witness graph is then entailed using the ontology and the applicable entailment regime,

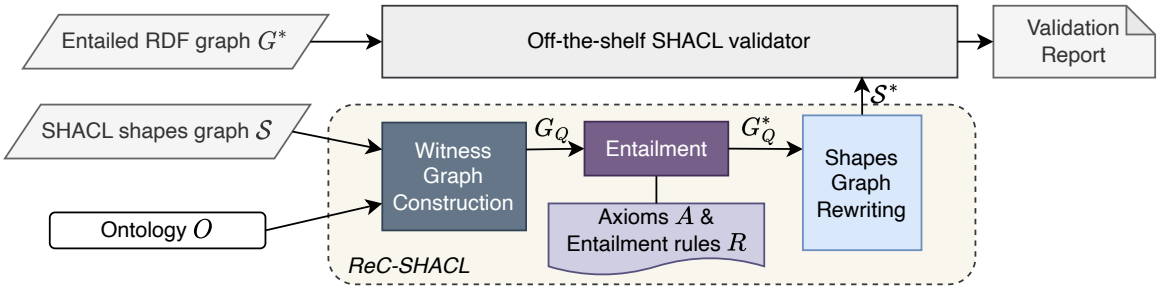


Figure 2: Overview of our approach ReC-SHACL.

producing the entailed witness graph G_Q^*. From G_Q^*, the extended set of permissible properties Q^+ is computed, and the original shapes graph is rewritten accordingly by adding new property shapes for the entailed properties and including universally entailed predicates in the ignored properties. The rewritten shapes graph, together with the independently entailed data graph, is then passed to an off-the-shelf SHACL validator, which produces the final validation report.

In the following, first we align syntactical and algebraic definitions of closed shapes in SHACL, which are essential to our work (Section 4.1). Next, we explain how ReC-SHACL exploits these definitions to produce rewritten shapes that account for entailment in the data graph (Section 4.2).

4.1. Extracting Permissible Properties from Closed Shapes

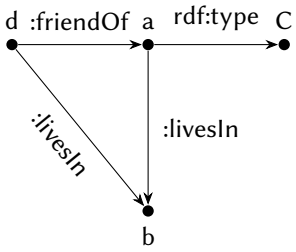
As shown in Figure 3b, the SHACL syntax utilizes a boolean `sh:closed` predicate to define closed shapes. However, the underlying algebra defines the expression $closed(Q)$ (Definition 6), which requires an explicit set of *permissible properties* Q. In practice, the validation engine dynamically constructs Q as the union of two sets, which we call the *declared edge set* and the *exempted edge set.*

Definition 9 (Permissible Properties in Closed Shapes)**.** *Given a SHACL closed shape φ, i.e., which includes the expression $closed(Q)$. The set of permissible properties $Q := P_{ign} \cup P_\varphi$, where the exempted edge set $P_{ign} \subseteq_{fin} P_G$ is given, and $P_\varphi \subseteq_{fin} P_G$ is the declared edge set of φ, defined as $P_\varphi := edge(\varphi)$.*

In practice, the exempted edge set P_{ign} corresponds to the properties listed in `sh:ignoredProperties`. P_{ign} contains properties permitted to appear on a node without being explicitly constrained or triggering a violation of the closed-world assumption. The declared edge set P_φ consists of properties that occur in `sh:path` expressions in the shape. P_φ is constructed recursively over the paths and shape structure, as defined in the following.

Definition 10 (Declared Edge Set)**.** *Let φ_s be a SHACL shape to be evaluated over a data graph G. The declared edge set (denoted as P_φ) is the set of predicates computed by the recursive function $edge(\cdot)$, defined over finite shape expressions. For a given simple path p, we define $edge(p) = \{p\}$. For shape φ, $edge(\varphi)$ is defined as follows:*

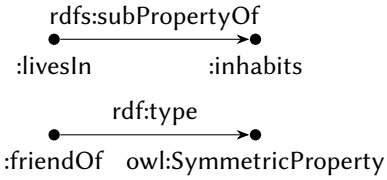
$$
\begin{aligned}
edge(\top) &= \emptyset. \\
edge(is(c)) &= \emptyset. \\
edge(test(E)) &= \emptyset. \\
edge(eq(p, p')) &= edge(p) \cup edge(p'). \\
edge(disj(p, p')) &= edge(p) \cup edge(p'). \\
edge(\neg\varphi) &= edge(\varphi). \\
edge(\varphi \wedge \varphi') &= edge(\varphi) \cup edge(\varphi'). \\
edge(\varphi \vee \varphi') &= edge(\varphi) \cup edge(\varphi'). \\
edge(\exists^{\geq n} p.\varphi) &= edge(p) \cup edge(\varphi).
\end{aligned}
$$



(a) Data graph

```
:s1 a sh:NodeShape ;
    sh:targetClass :C ;
    sh:closed true ;
    sh:property [
        sh:path :livesIn ;
        sh:minCount 1;
    ] ;
    sh:ignoredProperties (rdf:type) .
```

(b) Example SHACL closed shape



(c) Ontology

Figure 3: Example: (a) data graph, (b) SHACL closed shape, (c) ontology.

$$edge(\exists^{\leq n} p.\varphi) = edge(p) \cup edge(\varphi).$$

where $\top$ is the trivial constraint, $c \in V_G$ is a constant, E is a condition (e.g., datatype, class, value comparison, regular expression, etc.), $p, p' \in P_G$, φ, φ' are SHACL shapes, n is a natural number.

Example: computing permissible properties in a shape To validate the SHACL closed constraint, we need to retrieve the set of permissible properties Q. To this end, we perform a shape-based traversal of the given node shape using Definition 10:

$$\begin{aligned} P_\varphi &= edge(\varphi) \\ &= edge(\exists^{\geq 1} p.\top) \\ &= edge(p) \cup edge(\top) \\ &= \{\text{:livesIn}\} \cup \emptyset \\ &= \{\text{:livesIn}\}. \end{aligned} \qquad P_{ign} = \{\text{rdf:type}\}.$$

Thus, the set of permissible predicates is

$$Q = P_\varphi \cup P_{ign} = \{\text{:livesIn}\} \cup \{\text{rdf:type}\}.$$

Note that the closed constraint only restricts predicates outgoing from the focus node. Incoming edges (e.g., :friendOf) do not affect the validation result.

Definition 11 (SHACL Closed Shape Validation: Alternative Definition)**.** *Consider G a data graph. Let $props_G(v) := \{\, p \mid \exists w : (v, p, w) \in G \,\}$, i.e., all properties in a graph G given a node v as the subject. Let φ be a closed SHACL shape with $closed(Q)$. The focus node v conforms to φ if it holds that:*

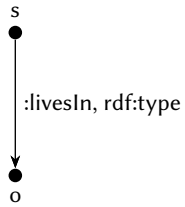
$$G, v \models closed(Q) \iff props_G(v) \subseteq Q.$$

Lemma 12. *Definition 6 and Definition 11 are equivalent.*

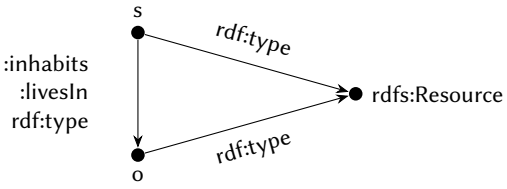
In the data graph in Figure 3a, we have $props_G(a) = \{\text{rdf:type}, \text{:livesIn}\}$. The node shape :s1 allows the set of predicates $Q = \{\text{rdf:type}, \text{:livesIn}\}$. Therefore, $props_G(a) \subseteq Q$, and the focus node a conforms to the node shape.

4.2. Shape Rewriting: Extending the Set of Permissible Predicates to Cope with Entailment

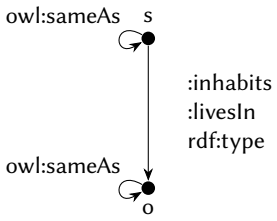
As shown in the previous section, entailment may introduce additional predicates in $props_{G^*}(v)$. This may affect the validation result, since the closed constraint requires $props_G(v) \subseteq Q$. Entailment of the



(a) Witness graph G_Q for properties in Q



(b) Entailed witness graph in G^*_Q after RDFS closure based on the ontology in Figure 3c



(c) Entailed witness graph in G^*_Q after OWL-LD closure based on the ontology in Figure 3c

Figure 4: Witness graph G_Q and the entailed triples given RDFS and OWL-LD entailment regimes.

data graph may introduce additional predicates for a focus node, which may lead to $props_{G^*}(v) \not\subseteq Q$. To determine which predicates may be introduced by entailment, we construct a witness graph based on the permissible properties Q.

Definition 13 (Witness Graph)**.** *A witness graph of a SHACL closed shape is a minimal data graph constructed from the permissible properties Q and two fresh IRIs s and o, such that*

$$G_Q := \bigcup_{q \in Q \setminus \{\textit{owl:sameAs}\}} \{(s, q, o)\}.$$

Not excluding owl:sameAs in the witness graph may lead to spurious predicates in Q^+. Assume the ontology contains the additional triple :inhabits owl:inverseOf :homeTo. This entails the triple o :homeTo s. If owl:sameAs were included in Q (e.g., as value of sh:path in a shape), then by Definition 13, for $q =$ owl:sameAs, the witness graph contains the triple s owl:sameAs o. This would entail the triple s :homeTo s. Consequently, :homeTo would be added to the set of permissible properties. Therefore, owl:sameAs is excluded from the construction of the witness graph. OWL entailment regimes may still entail the reflexive triple s owl:sameAs s.

We compute the closure of the witness graph using the ontology and an entailment regime R. Let $G^*_Q := cl_R(G_Q \cup O)$ be the closure of the graph G_Q (Definition 8). For RDFS, we get the additional triples as shown in Figure 4b. For OWL-LD, we get the triples shown in Figure 4c. $Q^+ := props_{G^*_Q}(s)$ is the permissible set of predicates after entailment. It follows that $Q \subseteq Q^+$ and therefore $G, v \models closed(Q) \Rightarrow G, v \models closed(Q^+)$.

Example Given the set of permissible predicates Q, we construct the witness graph G_Q defined in Definition 13. For every $q \in Q$, the witness graph contains the triples shown in Figure 4a.

Shape rewriting under RDFS entailment. To compute Q^+, we collect all predicates occurring in triples with subject s in Figure 4a together with the entailed triples in Figure 4b, yielding $Q^+ =$ {:livesIn, rdf:type, :inhabits}. When validating the focus node s, we check whether $props_{G^*}(s) \subseteq Q^+$. Since $props_{G^*}(s) =$ {:livesIn, rdf:type, :inhabits}, it follows that $props_{G^*}(s) \subseteq Q^+$. Therefore, the entailed data graph conforms to the rewritten shapes graph under RDFS entailment.

Shape rewriting under OWL-LD entailment. Q^+ is computed by collecting all predicates occurring in triples with subject s in the union of Figure 4a and Figure 4c. Thus, $Q^+ =$ {:livesIn, rdf:type, :inhabits, owl:sameAs}. However, the entailed data graph contains $props_{G^*}(s) =$ {:livesIn, rdf:type, :inhabits, :friendOf, owl:sameAs}. Since $props_{G^*}(s) \not\subseteq Q^+$, the focus node s violates the closed shape constraint under OWL-LD entailment. Furthermore, we can see that, under OWL-LD, we entail the property owl:sameAs rather than rdf:type for each node in the graph.

Ignored Properties Given an entailment regime, there may be predicates that are entailed for every entity, i.e.,

$$P'_{ign} := P_{ign} \cup \{\, p \mid \forall s,\ \exists G,\ \exists o : (s, p, o) \in cl_{\mathcal{R}}(G \cup O) \,\}.$$

Table 3
Dataset statistics. **Top**: data graph statistics (triples, subjects, predicates, objects). **Bottom**: shapes graph statistics (node shapes, property shapes, targeted classes, distinct `sh:path` properties, closed node shapes, constraints).

Data Graph	Triples	Subjects	Predicates	Objects
DBpedia (DB50)	534,696	134,303	2,888	140,855
DBpedia (DB100)	912,108	208,836	3,463	229,587

Shapes Graph	Node Shapes	Property Shapes	Targeted Classes	Targeted Props	Closed	Constraints
C_3	30	611	30	255	3	707
C_6	30	611	30	255	6	707
C_{15}	30	611	30	255	15	707
C_{30}	30	611	30	255	30	707

Since such predicates occur for every subject under R, they should not affect the outcome of closed-shape validation. Therefore, these predicates are added to the set of ignored properties. Under RDFS, rdf:type is added to the ignored properties, while under OWL-LD, owl:sameAs is added, provided that the property is not already defined as a path in a property shape.

5. Evaluation

In this section, we evaluate the proposed approach with respect to (i) the number of closed-shape violations and (ii) execution time. First, we investigate how reasoning affects closed-shape validation and whether it introduces additional violations. Next, we analyze how shapes-graph entailment affects the structural characteristics of the shapes graph, including the number of property shapes and targeted properties. We then evaluate validation results when entailed data graphs are validated against the entailed shapes graphs. We further compare our approach to the work, Re-SHACL, by Ke et al. [6], which addresses SHACL validation under reasoning. Finally, we measure the runtime of shapes-graph entailment and its impact on SHACL validation runtime.

RQ1 How does reasoning affect the number of violations in SHACL closed-shape validation?

RQ2 How does shapes-graph entailment alter the structural characteristics of SHACL shapes?

RQ3 How does our method influence closed-shape violations under the RDFS and OWL-LD entailment regimes?

RQ4 How does our method compare to Re-SHACL in terms of closed-shape violation counts under reasoning?

RQ5 What is the runtime overhead introduced by shapes-graph entailment during SHACL validation?

Datasets We evaluate the approach on two subsets of DBpedia. We use 30 shapes mined by Rabbani et al. [13]. For our closed shape experiments, we create four variants by randomly closing 3, 6, 15, and 30 shapes. These closed variants are constructed independently; i.e., shapes that are closed in one variant are not guaranteed to be closed in another. The resulting data and shapes graphs are summarized in Table 3. The table reports the number of node shapes, property shapes, targeted classes, distinct properties occurring in `sh:path` constraints, the number of closed node shapes, and the total number of constraints. All shapes graphs are identical except for the subset of node shapes marked as closed.

Test bench All tests were run on an Ubuntu 24.04 server, employing an AMD EPYC 9224 CPU (24c / 48t), 566 GiB RAM and a 7 TiB SATA SSD.

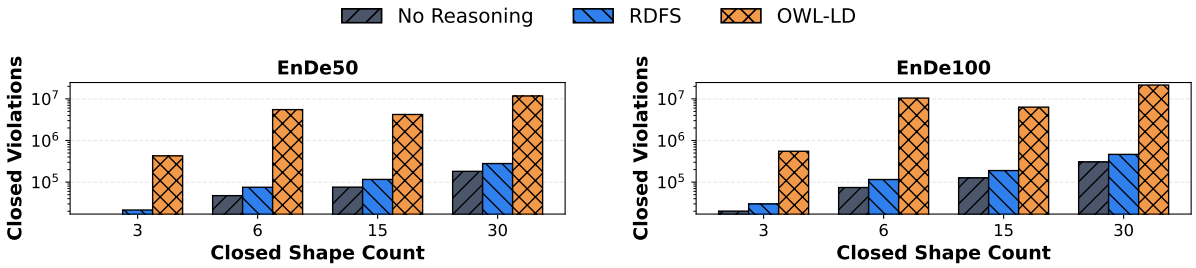


Figure 5: Impact of reasoning under different entailment regimes on the number of violations

Table 4
Shapes graph statistics for Shape_30 closed variants after shape rewriting under RDFS entailment. Columns report graph, number of property shapes, distinct `sh:path` properties, number of constraints, and ignored properties. Values in green indicate the increase due to rewriting.

Graph	Property Shapes	Targeted Props	Constraints	Ignored properties
C_3	611 (+18)	255 (+15)	707 (+0)	0 (+0)
C_6	611 (+10)	255 (+6)	707 (+0)	0 (+0)
C_{15}	611 (+10)	255 (+3)	707 (+0)	0 (+0)
C_{30}	611 (+45)	255 (+20)	707 (+0)	0 (+0)

Implementation Details The shape entailment was written in Python, using RDFLib to load the shapes graph and ontology. The data graph entailment was computed using rule-based reasoning in Apache Jena [14]. RDFS entailment used the built-in Jena RDFS reasoner. OWL-LD entailment used the Jena rule engine with a custom reduced rule set derived from the OWL 2 RL rules by retaining only the OWL-LD entailment rules.

5.1. Impact of Reasoning on Closed Shape Validation

Using an ontology to infer knowledge in an RDF graph may introduce additional triples, including properties that are not permitted by a closed shape. Consequently, more expressive entailment regimes may lead to additional closed-shape violations. To investigate the exact impact of reasoning on the violation count, we compare the number of closed-shape violations across different entailment regimes. Validation of the data graph without reasoning is included as a baseline. The same data graphs are then entailed under the RDFS and OWL-LD regimes and validated. Figure 5 shows the number of violations for the two data graphs. For RDFS, the increase is small in some cases, suggesting that many superproperty relations were already materialized in the data graph prior to reasoning. For OWL-LD, we observe a sharp increase in the number of closed-shape violations across all cases. A likely explanation is the presence of entailment rules involving owl:sameAs. These rules tend to increase the number of violations for two reasons: (1) violations are duplicated across nodes connected through owl:sameAs, and (2) owl:sameAs propagates properties between equivalent entities.

In summary, when implicit knowledge exists in the data graph, reasoning may increase the number of closed-constraint violations by materializing this knowledge (**RQ1**).

5.2. Effects of Shape Entailment on the Shapes Graph

To mitigate some of the effects of data-graph entailment on validation results, we compute shapes-graph entailment. This process may introduce additional permissible properties, which are added as new property shapes or included in sh:ignoredProperties. To analyze how the shapes graph changes during this process, we examine the number of property shapes, targeted properties, constraints, and ignored properties.

Table 5
Shapes graph statistics for Shape_30 closed variants after shape rewriting under OWL-LD entailment. Columns report graph, number of property shapes, distinct `sh:path` properties, number of constraints, and ignored properties. Values in green indicate the increase due to rewriting.

Graph	Property Shapes	Targeted Props	Constraints	Ignored properties
C_3	611 (+37)	255 (+31)	707 (+13)	0 (+1)
C_6	611 (+79)	255 (+53)	707 (+71)	0 (+1)
C_{15}	611 (+180)	255 (+76)	707 (+144)	0 (+1)
C_{30}	611 (+354)	255 (+119)	707 (+263)	0 (+1)

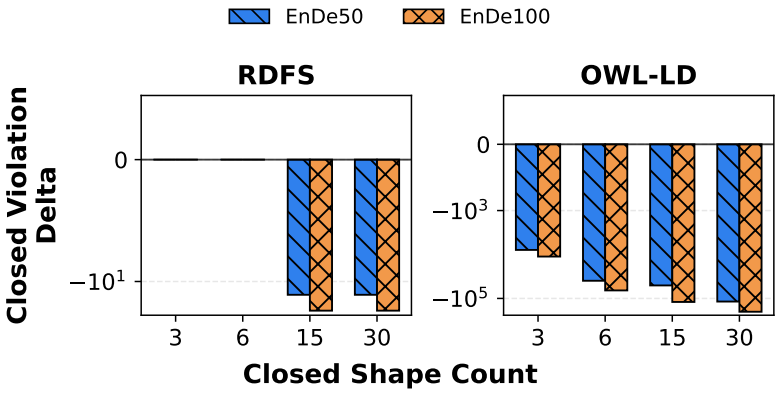


Figure 6: Impact of validating with the original shapes graph and the rewritten shapes graph under RDFS and OWL-LD entailment.

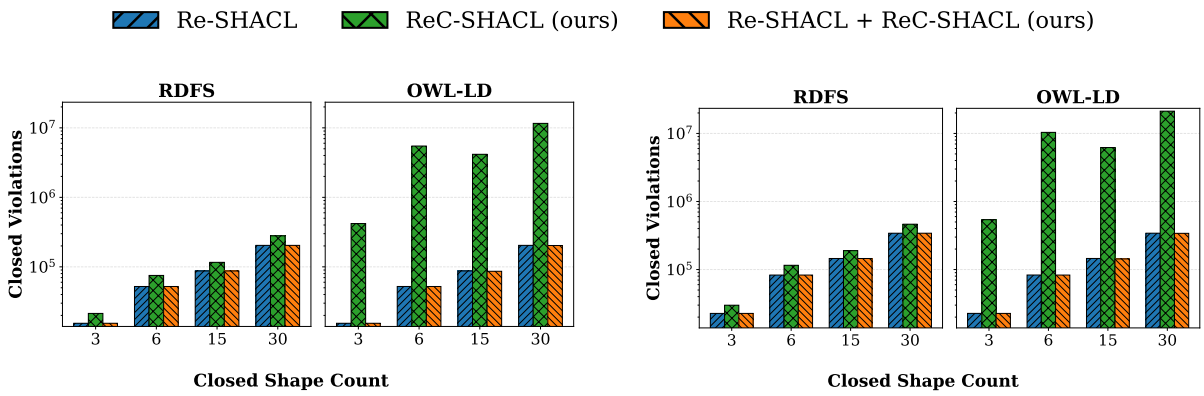
Table 4 shows these changes in the shapes graph after RDFS entailment. It shows an increase in both the property shape count and the number of targeted properties for each shapes graph. The property shape count increases more than the number of targeted properties, because multiple property shapes can target the same property. The number of constraints remains unchanged, as new property shapes are introduced for entailed properties without copying any constraints from the original property shapes.

The statistics shown in Table 5 reflect OWL-LD entailment of the shapes graph. OWL-LD contains owl:equivalentProperty and owl:sameAs, which may introduce new triples with entailed properties that can be substituted for the properties in the original triples. Therefore, these new triples must adhere to the same constraints, and the entire property shape (including constraints) is copied when a property is entailed through either. Hence, we see a non-zero increment in the number of constraints for OWL-LD entailment of the shapes graph. The ignored properties are only updated if the property, e.g., owl:sameAs, is not yet present in the node shape.

In summary, shapes-graph entailment adds additional property shapes and targeted properties to the shapes graph, potentially introducing more property shapes than targeted properties. For RDFS, the number of constraints remains unchanged, since new property shapes are added without constraints, while under OWL-LD, constraints are potentially copied from existing property shapes (**RQ2**).

5.3. Validation under RDFS and OWL-LD of the Shapes Graph

Previous experiments showed that closed-constraint violations increase when validating a data graph after reasoning. They also showed that the shapes-graph entailment introduces additional property shapes and targeted properties. In this experiment, we investigate the effect of validating an entailed data graph using an entailed shapes graph. More precisely, we measure the change in closed-constraint violation counts when validating with the entailed shapes graph instead of the original shapes graph. Figure 6 shows the results of this experiment, indicating the change in the number of closed-constraint



(a) Data graph: Ende50

(b) Data graph: Ende100

Figure 7: Comparing the effects of ReC-SHACL, Re-SHACL and both on the number of closed violations.

violations for different datasets and entailment regimes. For RDFS, we observe a trend towards a larger violation delta as more shapes are closed, i.e., more violations are resolved when more closed-shape constraints are present. For OWL-LD, the reduction in closed-constraint violations is substantially larger, with the larger data graph showing a slightly larger delta. The number of closed shapes appears to have a stronger impact on the delta than the size of the data graph. The large discrepancy between RDFS and OWL-LD can be explained by differences in shapes-graph entailment. Table 4 and Table 5 show a substantial difference in the number of entailed property shapes between the two regimes. A similar difference is likely present in the data graph entailment, where OWL-LD may entail significantly more triples.

In summary, under both entailment regimes, RDFS and OWL-LD, the number of closed-constraint violations decreases when using the entailed shapes graph. The benefit of our approach is particularly pronounced under OWL-LD entailment, likely due to the larger amount of entailed knowledge (**RQ3**).

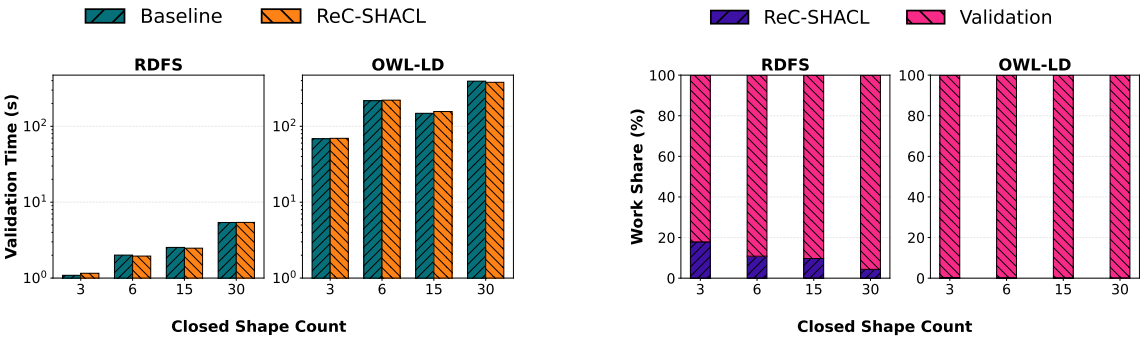
5.4. Comparing Different Approaches for SHACL Shape Reasoning

After evaluating validation with entailed shapes graphs, we compare our approach with another method addressing SHACL validation under reasoning. In particular, we compare our approach to Re-SHACL by Ke et al. [6], both individually and in combination with our entailed shapes graph. Performance is measured by the number of closed-constraint violations. Re-SHACL does not directly address closed shapes; however, it removes duplicate violations caused by owl:sameAs between equivalent nodes. Figure 7 shows the absolute number of closed-constraint violations for Re-SHACL with the original shapes, our entailed shapes, and the combination of Re-SHACL with our entailed shapes. The results show that combining Re-SHACL with our entailed shapes consistently yields the fewest violations. Applying Re-SHACL alone yields second-best results, indicating that a substantial portion of closed-constraint violations arises from duplicate violations.

In summary, while our approach reduces the number of violations, duplicate violations caused by owl:sameAs between focus nodes remain. Among the approaches considered, the combination of Re-SHACL with our entailed shapes yields the fewest violations (**RQ4**).

5.5. Investigating the Runtime Impact for Closed SHACL Shape Reasoning

Previous experiments showed the benefits of using rewritten shapes graphs when validating entailed data graphs. In practice, the applicability of our approach also depends on the runtime overhead it introduces. In this experiment, we analyze the runtime of our approach and its impact on SHACL validation time. We compare the cumulative runtime of shapes-graph rewriting and subsequent SHACL validation with the runtime of validating the original data graph.



(a) Total validation time

(b) Workload split between ReC-SHACL and validation time

Figure 8: Runtime and workload breakdown under different entailment regimes

Figure 8a compares the baseline (SHACL validation only) with SHACL validation using our entailed shapes graph. The total validation time is not noticeably affected by using our approach. For every shapes graph and under each entailment regime, the resulting validation time is almost identical to the baseline without shapes-graph entailment. Figure 8b shows the cumulative runtime of shapes-graph entailment and subsequent validation. For RDFS, shapes-graph entailment accounts for a small portion of the total runtime. This is because validation under RDFS reasoning is relatively fast. The percentage of runtime spent on entailment decreases as shape complexity increases, i.e., with a larger number of closed shapes. For OWL-LD, the percentage of total runtime spent on shapes-graph entailment is not visible in the plot (i.e., < 1%). Since OWL-LD entailment produces a larger data graph, validation also takes longer. However, this increase is not proportional to the shapes-graph entailment time, since shapes-graph entailment depends only on the shapes graph and the ontology. Therefore, the time spent on shapes-graph entailment is negligible compared to the total runtime.

In summary, under both RDFS and OWL-LD entailment regimes, shapes-graph entailment has only a small impact on total runtime. The percentage of runtime spent on entailment decreases as either the data graph grows or the complexity of the shapes graph increases (**RQ5**).

6. Related Work

Applying reasoning may infer new facts from a given data graph. These facts may include properties that are not mentioned in the SHACL shapes.

Ke et al. [6] improve the accuracy of SHACL validation under entailment regimes. They propose a pre-processing step that performs partial data graph entailment and shapes graph rewriting to reduce the number of spurious violations, while remaining compatible with existing SHACL validators. However, the method is not specifically designed for SHACL closed shapes. Therefore, spurious violations may still occur when validating closed shapes after reasoning.

Pareti et al. [15] study the interaction between a subset of SHACL and inference rules in Datalog. Given a set of SHACL constraints and inference rules, they determine whether an initially conforming graph remains conforming after applying the rules.

Several works study SHACL validation in the presence of ontologies. Savković et al. [7] propose a rewriting technique that compiles an OWL 2 QL ontology and a set of SHACL constraints into a stand-alone set of SHACL constraints, enabling validation over the data graph alone. They show that such a rewriting does not always exist for SHACL with full OWL 2 QL, and identify restricted fragments for which it does. Oudshoorn et al. [8] extend this approach to recursive SHACL with stratified negation, defining a validation semantics based on an austere canonical model and a corresponding rewriting algorithm. Both approaches ensure that the rewritten shapes graph preserves validation equivalence with respect to the ontology-entailed data graph. However, neither addresses how reasoning may

introduce new predicates that affect SHACL closed shapes.

In summary, existing work focuses on reducing spurious violations, identifying constraints that may be violated after reasoning, or incorporating implicit facts into the shapes graph. None of these approaches specifically addresses the behavior of SHACL closed shapes under entailment.

7. Conclusion

In this paper, we distinguish between entailment rules that cause spurious violations, those that cause sound violations, and those that have no effect on the closed constraint validation result. We presented ReC-SHACL, which extends the set of permissible properties of a SHACL-closed shape for validating an entailed data graph while mitigating the effects of spurious violations. We introduced the concept of a witness graph, which constructs triples from the permissible properties of a SHACL closed shape. Next, we compute the closure of the witness graph using a standard fixpoint entailment given an ontology. The resulting graph is used to extend the permissible properties with their logical consequences. Our experiments show that the method reduces the number of closed-constraint violations under both RDFS and OWL-LD entailment regimes. This reduction is due to the spurious violations being resolved through shape rewriting. Additionally, ReC-SHACL introduces only a small runtime overhead.

Several directions for future work remain. First, ReC-SHACL could be extended to support closed shapes with complex paths. Second, since entailment constitutes a form of graph update, closed-shape validation after entailment could be optimized using incremental validation techniques [16].

Declaration on Generative AI In preparing this work, the authors used Grammarly and GPT-5 to assist with spelling and grammar correction. The authors reviewed and edited all content and take full responsibility for the final manuscript.

References

[1] H. Knublauch, D. Kontokostas, Shapes Constraint Language (SHACL), W3C Recommendation, W3C, 2017. Https://www.w3.org/TR/2017/REC-shacl-20170720/.

[2] B. De Meester, P. Heyvaert, D. Arndt, A. Dimou, R. Verborgh, Rdf graph validation using rule-based reasoning, Semantic Web 12 (2020) 117–142. doi:10.3233/SW-200384.

[3] M. Figuera, P. D. Rohde, M.-E. Vidal, Trav-shacl: Efficiently validating networks of shacl constraints, in: Proceedings of the Web Conference 2021, WWW '21, Association for Computing Machinery, New York, NY, USA, 2021, p. 3337–3348. URL: https://doi.org/10.1145/3442381.3449877. doi:10.1145/3442381.3449877.

[4] D. Keuchela, N. Spicherb, J. Wangb, M. Völckerb, Y. Gongc, T. M. Desernob, Shacl-based report quality evaluation for health it-induced medication errors, in: MEDINFO 2021: One World, One Health—Global Partnership for Digital Innovation: Proceedings of the 18th World Congress on Medical and Health Informatics, volume 290, IOS Press, 2022, p. 414.

[5] A. Sommer, N. Car, pyshacl, 2023. URL: https://doi.org/10.5281/zenodo.10199105. doi:10.5281/zenodo.10199105.

[6] J. Ke, Z. Zacouris, M. Acosta, Efficient validation of shacl shapes with reasoning, Proceedings of the VLDB Endowment 17 (2024) 3589–3601.

[7] O. Savković, E. Kharlamov, S. Lamparter, Validation of shacl constraints over kgs with owl 2 ql ontologies via rewriting, in: P. Hitzler, M. Fernández, K. Janowicz, A. Zaveri, A. J. Gray, V. Lopez, A. Haller, K. Hammar (Eds.), The Semantic Web, Springer International Publishing, Cham, 2019, pp. 314–329.

[8] A. Oudshoorn, M. Ortiz, M. Šimkus, Shacl validation in the presence of ontologies: Semantics and rewriting techniques, Artificial Intelligence (2026) 104483.

[9] B. Glimm, A. Hogan, M. Krötzsch, A. Polleres, Owl: Yet to arrive on the web of data?, 2012. arXiv:1202.0984.

[10] S. Ahmetaj, I. Boneva, J. Hidders, K. Hose, M. Jakubowski, J. E. Labra Gayo, W. Martens, F. Mogavero, F. Murlak, C. Okulmus, et al., Common foundations for SHACL, ShEx, and PG-Schema, in: Proceedings of the ACM on Web Conference 2025, 2025, pp. 8–21.
[11] J. Corman, J. L. Reutter, O. Savković, Semantics and validation of recursive shacl, in: The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part I 17, Springer, 2018, pp. 318–336.
[12] A. Hogan, E. Blomqvist, M. Cochez, C. d'Amato, G. D. Melo, C. Gutierrez, S. Kirrane, J. E. L. Gayo, R. Navigli, S. Neumaier, et al., Knowledge graphs, ACM Computing Surveys (Csur) 54 (2021) 1–37.
[13] K. Rabbani, M. Lissandrini, K. Hose, Extraction of validating shapes from very large knowledge graphs, Proceedings of the VLDB Endowment. 16 (2023) 1023–1032.
[14] Apache Software Foundation, Apache Jena, https://jena.apache.org/, 2025. Accessed: 2025-07-31.
[15] P. Pareti, G. Konstantinidis, T. J. Norman, M. Şensoy, Shacl constraints with inference rules, in: The Semantic Web – ISWC 2019, Springer International Publishing, Cham, 2019, pp. 539–557.
[16] Z. G. Zacouris, J. Ke, M. Acosta, Upshacl: Targeted constraint validation for updates over knowledge graphs, in: The Semantic Web - ISWC, volume 16140 of *Lecture Notes in Computer Science*, Springer, 2025, pp. 122–139.