Efficient Validation of SHACL Shapes with Reasoning

Jin Ke
Technical University of Munich
Heilbronn, Germany
jin.ke@tum.de

Zenon Zacouris
Technical University of Munich
Heilbronn, Germany
zenon.zacouris@tum.de

Maribel Acosta
Technical University of Munich
Heilbronn, Germany
maribel.acosta@tum.de

ABSTRACT

As the usage of knowledge graphs (KGs) becomes more pervasive in practical applications, there is a burgeoning need for high-quality data. The SHApes Constraint Language (SHACL) allows for expressing certain types of quality constraints that define sub-structures and correct values in KGs modelled with RDF. Nevertheless, performing SHACL validation without entailment often yields one-sided outcomes, as it falls short of validating crucial implicit data encoded in the KG ontology. Current solutions that incorporate entailment into SHACL validation are inefficient, due to the time-intensive process of applying inference rules to the entire dataset. Moreover, applying entailment for SHACL validation can generate large amounts of redundant triples, exacerbating the validation workload and resulting in erroneous or redundant validation results. In light of these challenges, we propose Re-SHACL, an approach that combines targeted reasoning and entity merging techniques to generate a concise, consolidated RDF graph devoid of redundancy. Re-SHACL significantly reduces execution time and improves the accuracy of the validation reports. Our experiments demonstrate that Re-SHACL can be combined with state-of-the-art validators to deliver accurate validation reports efficiently.

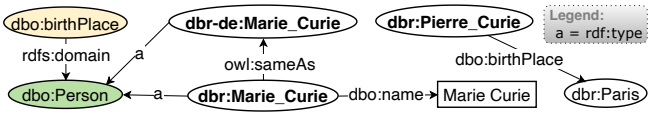
PVLDB Reference Format:
Jin Ke, Zenon Zacouris, and Maribel Acosta. Efficient Validation of SHACL Shapes with Reasoning. PVLDB, 17(11): 3589 - 3601, 2024.
doi:10.14778/3681954.3682023

PVLDB Artifact Availability:
The source code, data, and other artifacts have been made available at https://github.com/DE-TUM/Re-SHACL.

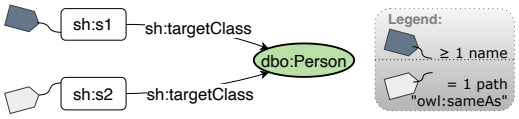
1 INTRODUCTION

SHACL (SHApes Constraint Language) [25] is a W3C Recommendation for validating knowledge graphs modelled with the Resource Description Framework (RDF). As a language for expressing constraints on RDF graphs, SHACL allows for defining the structure of RDF data, such as which properties must be present, which datatypes are allowed, and for a given property which values are permitted. SHACL validation provides a mechanism to ensure that the RDF graph follows a specified pattern, thereby improving data quality and interoperability [24, 30, 41, 42] and enabling the development of advanced automated applications [34].

Proceedings of the VLDB Endowment, Vol. 17, No. 11 ISSN 2150-8097.
doi:10.14778/3681954.3682023



(a) Data graph



(b) Shapes graph

Shape	Entity	w/o Reasoning	Reasoning
:s1	dbr:Pierre_Curie	✓	✗
:s1	dbr:Marie_Curie	✓	✓
:s1	dbr-de:Marie_Curie	✗	✓
:s2	dbr:Pierre_Curie	✗	✓
:s2	dbr:Marie_Curie	✓	✗
:s2	dbr-de:Marie_Curie	✗	✗

(c) Validation results. ✓ conformance, ✗ violation. Red denotes undesired result, green otherwise

Figure 1: Motivating examples

RDF graphs are typically enhanced with semantics, where the meaning of terms are formally described in an ontology using RDFS or OWL definitions. Despite this, only a few SHACL engines take the semantics of ontologies into account [31, 35] by combining SHACL validation with entailment. The incorporation of entailment enables SHACL engines to identify implicit relationships between entities, properties, and classes, thus achieving a thorough and accurate data validation. Yet, enabling entailment not only increases the validation runtime but can also introduce undesired behaviors. This is due to the reflexivity of certain properties in the entailment regime and the *non-Unique Named Assumption (nUNA)* followed in knowledge graphs. Under nUNA, semantically equivalent entities are represented with different identifiers, yet they are connected with the `owl:sameAs` property. To illustrate the effects of SHACL validation with entailment, consider the following examples.

Motivating Examples. Fig. 1 shows (a) an RDF graph, and (b) the SHACL constraints to be validated over each entity. Every person (`dbo:Person`) should conform to the constrains `:s1` and `:s2`. Shape `:s1` states that every person should have at least a value for the predicate `dbo:name`. Shape `:s2` states that every person should have exactly one value for the predicate `owl:sameAs`. To consider entailment, an OWL reasoner is executed over the RDF graph to materialize all implicit triples. The validation results without and with reasoning are shown in Fig. 1(c). For `:s1` without reasoning, the entity `dbr:Pierre_Curie` conforms to the shape, as it is not

a dbo:Person entity. However, this is interpreted as a false positive, as when considering the domain of dbo:birthPlace, it is entailed that this entity should be targeted by this shape. Therefore, with reasoning, dbr:Pierre_Curie violates :s1. This shows the importance of considering semantics to produce accurate results. However, some properties' reflexivity can produce spurious SHACL validations with reasoning when it is not handled properly. Without reasoning, the entity dbr:Pierre_Curie does not conform to :s2. This result is correct. However, with reasoning, due to the reflexivity of the owl:sameAs property, we entail the triple (dbr:Pierre_Curie, owl:sameAs, dbr:Pierre_Curie), for which dbr:Pierre_Curie now conforms to shape :s2. This result can be considered spurious or incorrect, as the conformance is due to redundant entailed data and not the actual data graph structure. Moreover, because of the nUNA, the violations found for dbr:Marie_Curie are also reported for dbr-de:Marie_Curie, which quickly inflates the validation reports with duplicate results.

Challenges. Incorporating entailment in the SHACL validation process imposes several challenges. First, entailment approaches demand significant computational resources to deduce implicit triples [20]. Second, when entailment is carried out with reasoners, these tend to generate *redundant data* for the validation. In the context of SHACL, redundant data can be, e.g., triples entailed from reflexive properties or duplicated triples due to the nUNA assumption, where the reasoner "copies" the triples from one entity to another equivalent entity. This redundant data again affects the validation process's runtime, as the graph size with inferred statements may increase drastically. Moreover, redundant data can also hinder the interpretation of violation reports due to duplicate violations generated under the nUNA assumption. This leads to poor scaling and difficulty locating the root cause of violations [31].

Proposed Solution. We present Re-SHACL, an approach that enhances the RDF graph (data graph) and constraints (shapes graph) before validation, considering the ontology in the data graph and an entailment regime. Instead of computing the closure of the data graph, Re-SHACL extracts relevant information from the shapes graph to identify the parts of the data graph to which the entailment is applied. To cope with the nUNA, Re-SHACL implements a merging mechanism, where semantically equivalent entities are consolidated into a unified representation. Because of this merging, Re-SHACL rewrites the shapes graph to replace the identifiers of the merged entities mentioned in the constraints. The combination of the targeted reasoning and entity merging allows Re-SHACL to obtain concise, semantically enhanced data graphs, which can be efficiently evaluated using any SHACL validator without entailment.

Contributions. Re-SHACL is novel in the following ways:

(1) incorporates the semantics defined in ontologies and entailment regimes into the SHACL validation process,
(2) is robust to duplicate and false violations due to nUNA and reflexive properties defined in RDFS and OWL,
(3) is efficient as it enhances the data and shapes graph in a targeted manner to avoid unnecessary operations,
(4) can be used with any state-of-the-art SHACL validator.

We conducted experiments using synthetic and real-world datasets. Our results show that Re-SHACL is orders of magnitude faster than existing approaches that support RDFS and OWL entailment.

2 PRELIMINARIES & PROBLEM STATEMENT

Knowledge Graphs and Ontologies. We define a knowledge graph (KG) following the RDF data model. A KG G is a set of triples (s, p, o), where s, o are nodes and p is a predicate that corresponds to a labelled, directed edge between s and o. In a KG, nodes or entities can be assigned to classes using the predicate rdf:type. Logical definitions about the entities, predicates, and classes in a KG can be provided using RDFS and OWL.

The RDF Schema (RDFS) [2] provides definitions for hierarchies of classes and properties with the predicates rdf:subClassOf and rdfs:subPropertyOf, respectively. It also supports the definition of domain (rdfs:domain) and range (rdfs:range) of predicates, to assert the classes of nodes in the subject and object positions of a triple, respectively. The RDFS entailment regime include entailment patterns [21] or rules to establish that these predicates are transitive (rules *rdfs5* and *rdfs11*) and reflexive (rules *rdfs6* and *rdfs10*). The RDFS entailment regime also allows to entail that entities connected to specific predicates and classes are also connected to super-predicates and super-classes (rules *rdfs7* and *rdfs9*).

The Web Ontology Language (OWL) [36] is a more expressive language than RDFS. OWL supports semantic equivalences between entities (owl:sameAs) and classes (owl:equivalentClass). Furthermore, with OWL it is possible to state that properties are (a)symmetric, (ir)reflexive, (inverse) functional, etc. The current OWL 2 [23] specification distinguishes between different profiles. The profile OWL 2 QL is designed to support query answering, yet, it does not support individual equality assertions (owl:sameAs) typically found in real-world KGs. The profile OWL 2 RL is tailored to rule-based reasoning. In comparison to OWL 2 QL, it supports individual equality assertions and defines the reflexivity, symmetry, and transitivity of the owl:sameAs predicate (rules *eq-ref*, *eq-sym*, *eq-trans*, respectively). Despite its higher expressivity (w.r.t. OWL 2 QL), OWL 2 RL still exhibits desirable computational guarantees, as it avoids non-deterministic reasoning (which occurs when constructs introduce existential variables during reasoning) with data complexity in PTime-complete [32]. Lastly, Glimm et al. [16] presents a subset of OWL 2 RL entitled OWL LD, which comprises OWL definitions that are typically encountered in real-world RDF KGs. Since OWL LD also supports individual equality assertions, it provides the same computational guarantees as OWL 2 RL.

SHACL Validation. SHACL validation requires two inputs formally defined as follows [7, 9]: the *data graph* G is an RDF KG to be validated, and the *shapes graph* $\mathcal{S}$ which is a set of shapes defined as a triple $(s, targ_s, \varphi_s)$, where s is a shape name, $targ_s$ is a (possibly empty) monadic query to a shape to retrieve the focus nodes, and φ_s is a constraint. In SHACL, constraints are defined as follows [9]:

$$\varphi_s ::= \top \mid s \mid I \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \geq_n r.\varphi \mid EQ(r1, r2)$$

where s is a shape name, I is a condition (e.g., datatype, class, value comparison, regular expression, etc.), $r, r1, r2$ are expressions of predicates (or property paths), $n \in \mathbb{N}^+$, and $\varphi, \varphi_1, \varphi_2$ are SHACL constraints. We say that x *occurs in* φ_s if x is any of the elements defined above in φ_s, e.g., if φ_s is $\geq_1 r.\varphi$, then it holds that r, φ, as well as $r.\varphi$ *occurs in* φ_s. In the remainder, we abuse notation and write $s \in \mathcal{S}$ to denote that $(s, targ_s, \varphi_s) \in \mathcal{S}$. We also use the syntantic sugar $\leq_n r.\varphi$ for $\neg(\geq_{n+1} r.\varphi)$ and $r.\varphi$ for $\leq_0 \neg r.\varphi$.

We illustrate this notation using a shape from the example from Fig. 1. For the shape $\texttt{:s1} \in \mathcal{S}$, $targ_{\texttt{:s1}} = \{\texttt{dbr-de:Marie\_Curie}, \texttt{dbr:Marie\_Curie}\}$, and $\varphi_{\texttt{:s1}}$ corresponds to $\geq_1 \texttt{dbo:name}.\top$ which states that each person must have at least one name.

We denote the result of validating a shape s over a node v in G as $[\![\varphi_s]\!]^{v,G}$, where the latter is recursively defined by Corman et al. [6]. If v conforms to the shape s, the result of the validation is 1; otherwise, it is considered a violation. We define the validation of $\mathcal{S}$ over G, denoted $[\![\mathcal{S}]\!]^G$, as $\{(v, s, [\![\varphi_s]\!]^{v,G}) \mid s \in \mathcal{S},\ v \in G, v \in targ_s\}$.

SHACL validation with entailment, denoted $[\![\mathcal{S}]\!]^{G,\mathcal{E}}$, checks the conformance of the data graph G following an entailment regime $\mathcal{E}$ to a shapes graph $\mathcal{S}$. In this case, the implicit information in G entailed by $\mathcal{E}$ is considered in the validation process.

Problem Statement. This work focuses on methods to support efficient SHACL validation under entailment, i.e., $[\![\mathcal{S}]\!]^{G,\mathcal{E}}$. In practice, the computation of the validation result is executed by a validation strategy, which we denote by $\mathbb{S}$ and the universe of strategies by $\mathfrak{S}$. To generate a validation result, a strategy $\mathbb{S}$ is executed on a given data graph G under the entailment regime $\mathcal{E}$ against the given shapes graph $\mathcal{S}$, which we denote by $\mathbb{S}(\mathcal{S}, G, \mathcal{E})$. Then, we use the runtime of the strategy $time\big(\mathbb{S}(\mathcal{S}, G, \mathcal{E})\big)$ to evaluate its efficiency.

Definition 2.1 (Problem Statement). Given a data graph G, a shapes graph $\mathcal{S}$, and an entailment regime $\mathcal{E}$. The efficient SHACL validation with entailment problem is defined as devising a strategy $\mathbb{S} \in \mathfrak{S}$ s.t.

$$\arg\min_{\mathbb{S} \in \mathfrak{S}} \Big(time\big(\mathbb{S}(\mathcal{S}, G, \mathcal{E})\big)\Big), \text{ subject to } \mathbb{S}(\mathcal{S}, G, \mathcal{E}) = [\![\mathcal{S}]\!]^{G,\mathcal{E}}.$$

3 RELATED WORK

Since entailment regimes incorporate rules, we first analyse rule-based validation approaches over KGs. Second, we analyse the state-of-the-art SHACL validators over KGs that do not incorporate entailment regimes. Lastly, we briefly revisit related work for relational databases and semi-structured sources.

Rule-based validation over KGs. Several approaches support the inclusion of rules for SHACL validation. These rules can be defined in entailment regimes (e.g., RDFS, OWL) or in (domain-specific) inference rules. These approaches follow mainly two types of validation strategies, which extend the data graph or the shapes graph to incorporate the rules. Then, the actual validation process can be carried out with an off-the-shelf validator using the extended structures. One of these strategies relies on materializing the closure G^* of the data graph G for the set of rules; this strategy can be applied only in the presence of rules that do no introduce the existence of nodes that are not explicit in the original graph G. This strategy is implemented by pySHACL [40], which currently supports the entailment regimes RDFS [22] and OWL 2 RL [33]. The main drawbacks of this strategy are that G^* can be expensive to compute, and it contains a large number of redundant, inferred triples, exacerbating the validation performance. A second strategy relies on rewriting the shapes graph $\mathcal{S}$ into a graph $\mathcal{S}'$, which extends the shapes to incorporate the information encoded in the inference rules. The works by Ahmetaj et al. [1] and Savković et al. [39] provide correct shapes re-writings to incorporate the OWL 2 QL entailment regime into SHACL validation. However, these works do not handle the semantics of the `owl:sameAs` predicate, as more expressive fragments of OWL 2 are required for this. Validatrr [31] also applies shapes rewriting to support certain OWL features during SHACL validation. Yet, Validatrr does not provide theoretical guarantees, therefore, some OWL constructs may generate an exponential number of possible rewritings, as shown for the analogous problem of ontology-based data access [18]. To mitigate the issues of the two aforementioned strategies, a hybrid strategy can be applied. For instance, the solution by Pareti et al. [34, 35] extends the data graph schema and rewrites the queries used in $targ_s$ and the constraints φ_s (cf. Sect. 2) to incorporate inference rules. Our approach also follows a hybrid strategy, yet it differs in two main aspects: (i) it is tailored to entailment regimes and mitigates the undesired results due to reflexive properties defined in RDFS and OWL and due to the non-Unique Named Assumption followed in RDF graphs; and (ii) our shapes graph rewritings correspond to simple substitutions, which makes our solution very efficient.

Validation without entailment over KGs. Other approaches [6, 14] focus on the problem of efficient SHACL validation directly over the data graph, without considering the semantics of ontologies and entailment regimes. SHACL2SPARQL [6] supports recursive SHACL constraints [9]; its validation strategy consists in translating the SHACL shapes into SPARQL queries that can can be executed over the data graph to detect violations. Trav-SHACL [14] minimizes the number of operations during validation by implementing traversing and query rewriting techniques over the shapes graph to efficiently evaluate a specific type of constraints expressed in SHACL. In contrast to these approaches, our solution enhances the shapes and data graph with entailed triples, before the actual validation is carried out. Therefore, SHACL2SPARQL and Trav-SHACL, or any other state-of-the-art SHACL validator can be used in combination with our proposed solution to enable validation with entailment.

Validation over other data models. The problem of constraint validation has been widely investigated in the context of data quality [10] over the relational model and XML. The techniques for relational databases focus on the discovery of different constraints types, i.e., functional dependencies (FDs), conditional FDs, pattern FDs, edit rules, denial constraints (e.g. [5, 11, 13, 37, 44]). Most of these works present techniques to efficiently validate whether relations conform to the constraints. Other solutions compute repairs to constraint violations (e.g. [3, 26, 29]). In comparison to these works, we do not present a validation or repair engine, but a solution to incorporate into the data graph and the constraints the implicit knowledge generated during reasoning under the RDFS and OWL entailment regimes. Also, the aspect of reasoning in our work is not the same as the notion of "reasoning over FDs" [11] in databases. The latter aims at determining relationships between FDs to study satisfiability, consistency, and implication problems of the constraints. Several works have focused on the validation of semi-structured data models [27] and XML documents [4, 12] with different constraint types, e.g., keys, foreign keys, integrity constraints, FDs, and embedded dependencies. Yet, the difference and novelty of our work is the application of targeted reasoning to the RDF graph to incorporate the semantics of ontologies using entailment regimes.

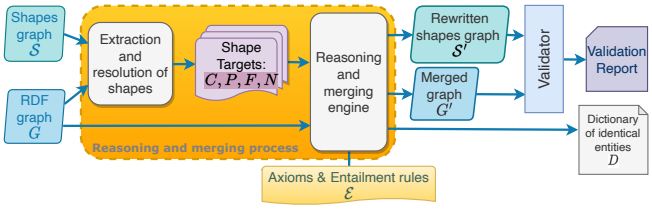


Figure 2: Components view of the shape-based reasoning and merging process of Re-SHACL

4 OUR APPROACH: RE-SHACL

We propose Re-SHACL, an approach based on materialization and shape rewriting that considers the entities, classes, and properties that occur in the shapes to carry out targeted reasoning and merging to perform efficient SHACL validation with entailment.

Fig. 2 shows the process of SHACL validation with entailment implemented by Re-SHACL. First, our approach extracts from the shapes graph $\mathcal{S}$ the focus nodes and property paths to guide the (targeted) reasoning and merging engine. This engine performs entailment on elements in the data graph G involved in the shape targets and executes merging operations solely on nodes that occur as focus nodes. The output of this engine is an RDF graph G' enhanced with entailed triples, denoted as a *merged graph*, wherein focus nodes with the same semantics are merged into a unified representation. Equivalence relations between merged nodes are stored in a dictionary to prevent semantics about equivalent entities from being lost during node merging, facilitating the identification of violation sources if needed. Moreover, the shapes graph is rewritten into another graph $\mathcal{S}'$ to align with the merged graph while merging target nodes. Finally, the validator is executed using G' and $\mathcal{S}'$ to produce the validation report. The validator in Re-SHACL is an off-the-shelf component, and can be exchanged by any state-of-the-art SHACL validator (e.g. [6, 14, 40]).

4.1 Extraction and Resolution of Shapes

The initial step of Re-SHACL is to extract meaningful information from the shapes graph $\mathcal{S}$. The SHACL vocabulary [25] distinguishes between two types of targets in shapes: classes (using the predicate `sh:targetClass`), and entities (using the predicate `sh:targetNode`). Therefore, in adition to $targ_s$ (cf.), we use $targClass_s$ and $targNode_s$ to incorporate this distinction between targets. Subsequently, Re-SHACL extracts relevant information from $\mathcal{S}$ and generates four initial structures as follows:

Set of Target Classes C: All the classes targeted in a shape (obtained with $targClass_s$) or mentioned in the shape constraints.

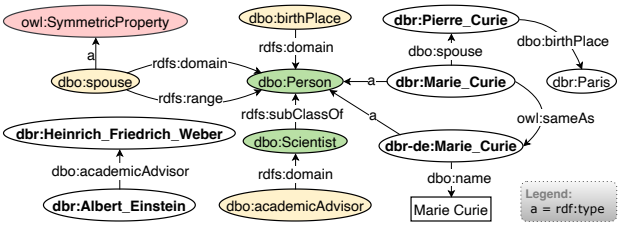
$$C := \{c \mid s \in \mathcal{S}, c \in targClass_s\} \cup \{c \mid s \in \mathcal{S}, c \text{ is a class that occurs in } \varphi_s\}$$

Set of Target Nodes N: Contains the nodes from the data graph that are targeted in the shapes obtained using $targNode_s$.

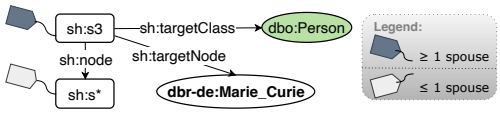
$$N := \{x \mid s \in \mathcal{S}, x \in targNode_s\}$$

Set of Target Properties P: Contains all predicates specified in a property path of a shape constraint.

$$P := \{r \mid s \in \mathcal{S}, r \text{ occurs in } \varphi_s\}$$



(a) Data graph G



(b) Shapes graph $\mathcal{S}$

Figure 3: Running example

Set of Focus Nodes F: Contains the focus nodes in each shape s (obtained with $targ_s$), and also the nodes from G that are objects of property paths in s that are validated using another shape s'.

$$F := \{x \mid s \in \mathcal{S}, x \in targ_s\} \cup \{x \mid s, s' \in \mathcal{S}, (\exists r, y) \text{ s.t. } r.s' \text{ occurs in } \varphi_s, (y, r, x) \in G\}$$

The output of this component is the shape targets combining C, P, F, N, which are used by the reasoning and merging engine.

Example 4.1. For the example shown in Fig. 3, in the initial state, the set $C :=$ {`dbo:Person`} because the node shape `:s*` has no target declaration and `:ss` declares `dbo:Person`. The set P includes only the `dbo:spouse` property, as `:s3` and `:s*` identify it as the single target property. $N :=$ {`dbr-de:Marie_Curie`} since shape `:ss` defines this specific target node. Finally, the set F comprises entities `dbr-de:Marie_Curie`, `dbo:Marie_Curie` and `dbo:Pierre_Curie`. Here, `dbo:Marie_Curie` stands out as the sole explicitly defined instance of `dbo:Person` in G. Meanwhile, `dbo:Pierre_Curie` represents a node identified through a property path $r :=$ `dbo:spouse` within the property shape `:s3` in `:ss`, which passed to the node shape `:s*` due to `:ss` referencing `:s*`, i.e., $r.$`:s*` occurs in $\varphi_{:\texttt{ss}}$.

4.2 Reasoning and Merging Engine

This component prepares the data and shapes graphs for the validator. The reasoning engine (§4.2.1) entails triples based on the shape targets. The merging engine (§4.2.2) computes a unified representation of focus nodes that contain semantically equivalent entities (defined via `owl:sameAs`). Lastly, Re-SHACL rewrites the shapes graph (§4.2.3) to replace the occurrences of merged focus nodes with the identifier of the unified representations.

4.2.1 Shape-based reasoning. The Re-SHACL reasoning engine enhances the data graph G with entailed triples relevant for the validation process. Before the reasoning starts, Re-SHACL builds a dictionary D. Each key in D is an entity that corresponds to a unified representation of a set of entities connected with `owl:sameAs`. The D keys are initialized with each focus node in F with an empty set of values. For the example in Fig. 3, we have $D :=$ {`dbo:Marie_Curie` : $\emptyset$, `dbo:Pierre_Curie` : $\emptyset$, `dbo-de:Marie_Curie` : $\emptyset$}.

Algorithm 1: Shape-based Class Reasoning

```
Data: RDF graph G, target classes C, focus nodes F, and entity dictionary D
1  for all c ∈ C do
2      NewNodes ← ∅
       /* Highlighted lines only executed during OWL reasoning */
3      EC ← φ_EC(c)                                  // Set of equivalent classes
4      T ← φ_ECT(c)                  // Triples asserting equivalent classes
5      EC ← EC − {c}
6      G ← (G − T) ∪ {(c,rdfs:subClassOf,x),(x,rdfs:subClassOf,c) | x ∈ EC}
                                                     // RULE scm-eqc1
7
8      SC ← φ_SC(c)                                  // Set of subclasses
9      SC ← SC − {c}
10     for sc ∈ SC do
11         NewNodes ← NewNodes ∪ {i | (i,rdf:type,sc) ∈ G}
12         for all p such that (p,rdfs:domain,sc) ∈ G do        // RULE scm-dom1
13             G ← G ∪ {(p,rdfs:domain,c)}
14         for all p' such that (p',rdfs:range,sc) ∈ G do       // scm-rng1
15             G ← G ∪ {(p',rdfs:range,c)}
16     for p1 ∈ {p | (p,rdfs:domain,c) ∈ G} do                  // RULE scm-dom2
17         for ep ∈ φ_EP(p1) do
18             G ← G ∪ {(ep,rdfs:domain,c)}
19         for sp ∈ φ_SP(p1) do
20             G ← G ∪ {(sp,rdfs:domain,c)}
21             for sp_ep ∈ φ_EP(sp) do
22                 G ← G ∪ {(sp_ep,rdfs:domain,c)}
23     for p2 ∈ {p | (p,rdfs:range,c) ∈ G} do                   // RULE scm-rng2
24         for ep ∈ φ_EP(p2) do
25             G ← G ∪ {(ep,rdfs:range,c)}
26         for sp ∈ φ_SP(p2) do
27             G ← G ∪ {(sp,rdfs:range,c)}
28             for sp_ep ∈ φ_EP(sp) do
29                 G ← G ∪ {(sp_ep,rdfs:range,c)}
30     NewNodes ← NewNodes ∪ {o | {(s,p,o),(p,rdfs:range,c)} ∈ G} ∪ {s |
         {(s,p,o),(p,rdfs:domain,c)} ∈ G}
                                   // RULE prp-dom,prp-rng,rdfs2,rdfs3
31     for x ∈ NewNodes do                  // RULE rdfs9,cax-sco,cax-eqc1,cax-eqc2
32         if x ∉ F then
33             F ← F ∪ {x}
34             D ← D ∪ {x → ∅}
35             G ← G ∪ {(x,rdf:type,c)}
36     C ← C ∪ EC                           // Only executed during OWL reasoning
       C ← C ∪ SC − {c}
37 return G, C, F, D
```

Then, the Re-SHACL reasoning process is performed considering the shape targets in two phases: Phase 1 performs class-related reasoning on all target classes in set C, while Phase 2 performs property-related inferences for each target property in P. Currently, Re-SHACL supports the RDFS [22] and OWL LD [17] entailment regimes. OWL LD is chosen, as it shows a good tradeoff between expressivity and performance, while supporting features that are frequently present in real-world RDF graphs [17]. Furthermore, since OWL LD is designed to be compatible with RDFS, i.e., Re-SHACL can seamlessly support RDFS reasoning. Next, we describe how the RDFS and OWL LD entailment rules are applied to the shape targets in each phase to enhance G and obtain G'. For simplicity, in the remainder, we refer to G' as simply G.

Phase 1: Class reasoning. This phase is shown in Alg. 1 and starts by processing the equivalence classes and subclasses of target classes in the set C. Let the target class currently being processed c (line 1). Equivalent classes specified by `owl:equivalentClass` of c in the data graph are identified via the function ϕ_{EC} (line 3). The function ϕ_{ECT} is invoked to retrieve all triples that assert equivalent class relationships associated with the specified input classes (line 4). Subsequently, the *scm-eqc1* rule (i.e., class antisymmetry) infers that the equivalent classes to c and c are subclasses of each other. After this process, these triples are removed from the graph G to avoid redundancy during subsequent processing stages (line 6). This elimination does not entail any loss of information, as the representation of equivalence relations of two classes can be equally expressed through new information regarding their subclasses. The function ϕ_{SC} (line 7) systematically extracts all subclasses associated with a given class asserted with the `rdfs:subClassOf` predicate. ϕ_{SC} accounts for the transitivity of the subclass relationships.

For the subclasses of c, consideration is also required for reasoning based on rules *scm-dom1* and *scm-rng1* since both of these rules have the potential to identify other entities that also become focus nodes for validation. The rules *scm-dom1* and *scm-rng1* entail that the domain (lines 11-12) and range (lines 13-14) definitions in subclasses also apply to their superclasses. Then, the engine addresses the rules *scm-dom2* and *scm-rng2*; these rules entail that the domain (lines 15-21) and range (lines 22-28) definitions in subproperties also apply to their superproperties. These rules are applied to all equivalent properties (using ϕ_{EP}) and subproperties (using ϕ_{SP}) of the processed property, accounting for the transitivity of the `owl:equivalentProperty` and `rdfs:subPropertyOf` predicates. Subsequently, the algorithm extends G with all the information about equivalent properties, subproperties at different hierarchy levels in the ontology, and including properties whose domain or range type is also identified as c. Based on these properties, the NewNodes are extended to incorporate new target entities identified via reasoning on the ontology predicates in G.

Lastly, rules *prp-dom* and *prp-rng* (corresponding to rules *rdfs2* and *rdfs3* in the RDFS entailment regime) allow for inferring additional entities for the target class c based on their domain and range, respectively. These entities are also incorporated into F and recorded in the dictionary D as a key with an empty set of values. The algorithm adds the newly identified types for these entities to G, with the application of the rules *rdfs9* and *cax-sco* for the target class c; these rules are identical and assert that entities of subclasses also belong to the corresponding superclasses. The algorithm also applies rules *cax-eqc1* and *cax-eqc2* (extensionality) to entail that entities of a class c also belong to the equivalent classes of c.

Example 4.2. Consider the data graph G from Fig. 3 and the class `c := dbo:Person`. Initially, `NewNodes := ∅` (line 2). In G, there are no equivalent class to c, but there is a subclass of it, namely `dbo:Scientist`. Therefore, $SC :=$ {`dbo:Scientist`} (lines 7 to 8), which directly triggers the for loop in line 9. For `sc := dbo:Scientist`, the set NewNodes remains empty as there are no instances of sc in G (line 10). As noted in lines 11-12, the triple (`dbo:academicAdvisor,rdfs:domain, dbo:Person`) can be inferred from the given triple (`dbo:academicAdvisor,rdfs:domain, dbo:Scientist`). Since G does not match the antecedents for *scm-dom2* and *scm-rng2*, we come directly to line 29, where, using the triples (`dbo:spouse, rdfs:domain, dbo:Person`), (`dbo:spouse, rdfs:range, dbo:Person`), and the previously entailed triple, the entities `dbo:Albert_Einstein`, `dbo:Pierre_Curie` as well as `dbo:Marie_Curie` are added to the set NewNodes. This set now contains the new focus nodes that are inferred during the reasoning process. At line 30, the algorithm adds these new focus nodes to F

Algorithm 2: Shape-based Property Reasoning

```
   Data: RDF graph G, target properties P
1  for all fp ∈ P do
2      while exists subproperty of fp in G do
3          SP ← φ_SP(fp)
4          for each p' ∈ SP − {fp} do
               /* Highlighted lines are only executed during OWL          */
5              if (fp,rdfs:subPropertyOf,p') ∈ G' then        // RULE scm-eqp2
6                  G ← G ∪ {(fp,owl:equivalentProperty,p')}
7
8              else
                   // RULE rdfs5,scm-spo,scm-dom2,scm-rng2,prp-spo1,rdfs7
9                  for all subproperties p" of p' with p" ≠ fp do
10                     G ← G ∪ {(p",rdfs:subPropertyOf,fp)}
11                 for all classes c such that (fp,rdfs:domain,c) ∈ G do
12                     G ← G ∪ {(p',rdfs:domain,c)}
13                 for all classes c' such that (fp,rdfs:range,c') ∈ G do
14                     G ← G ∪ {(p',rdfs:range,c)}
                                                  // RULE prp-spo1 or rdfs7
15                 for all pairs of x, y such that (x,p',y) ∈ G do
16                     G ← G ∪ {(x,fp,y)}
17             G ← G − {(p',rdfs:subPropertyOf,fp)}
       /* Highlighted lines are only executed during OWL reasoning       */
18     while exists equivalent property of fp in G do
19         T' ← φ_EPT(fp)
20         EP ← φ_EP(fp)
21         for all sp ∈ EP − {fp} do        // RULE eq-rep-p, prp-eqp1,prp-eqp2
22             G ← G ∪ {(fp,p,o) | ∀(sp,p,o) ∈ G} ∪ {(s,p,fp) | ∀(s,p,sp) ∈
                   G} ∪ {(s,fp,o) | ∀(s,sp,o) ∈ G}
23         G ← G − T'
24     for each inverse property ip of fp do                // RULE prp-inv1,prp-inv2
25         for all (x,fp,y) ∈ G do
26             G ← G ∪ {(y,ip,x)}
27         for all (x',ip,y') ∈ G do
28             G ← G ∪ {(y',fp,x')}
29     if (fp,a,owl:SymmetricProperty) ∈ G then                // RULE prp-symp
30         for all (x,fp,y) ∈ G do
31             G ← G ∪ {(y,fp,x)}
32     if (fp,a,owl:TransitiveProperty) ∈ G then               // RULE prp-trp
33         for all {(x,fp,y),(y,fp,z)} ∈ G do
34             G ← G ∪ {(x,fp,z)}
35     if (fp,a,owl:FunctionalProperty) ∈ G then               // RULE prp-fp
36         for all {(x,fp,y1),(x,fp,y2)} ∈ G do
37             G ← G ∪ {(y1,owl:sameAs,y2)}
38     if (fp,a,owl:InverseFunctionalProperty) ∈ G then        // RULE prp-ifp
39         for all {(x1,fp,y),(x2,fp,y)} ∈ G do
40             G ← G ∪ {(x1,owl:sameAs,x2)}
41
42 return G
```

and records them in the dictionary D. Finally the algorithm updates G with the class memberships inferred for the new focus nodes.

Alg. 1 is executed until no new triples are entailed. The entire class-centric reasoning processing delineated above is confined to the target classes and significantly aids in identifying implicit focus nodes within the data graph. In the context of SHACL, the shape uses the target declaration to select the focus nodes among all the nodes in the data graph. This ensures that alterations to classes unrelated to the specified target class do not impact the target declaration or focus nodes. In other words, irrespective of whether reasoning is applied to these unrelated classes, the validation outcome does not change. By pruning non-targeted classes during reasoning, Re-SHACL aims to achieve high efficiency.

Phase 2: Property reasoning. This phase is shown in Alg. 2 and processes the property paths P indicated for validation in the shapes graph, herein referred to as target paths. Let us consider `tp` the target path under processing. The function ϕ_{SP} retrieves the subproperties of `tp` (line 3). If there exists a mutual subproperty relationship between `tp` and a subproperty, the equivalence between these two can be established with the rule *scm-eqp2* (lines 5–6). Otherwise, if a property is a proper subproperty of `tp` (i.e., not equivalent property), the algorithm proceeds to entail the triples according to the rules *prp-spo1* (or *rdfs7*), that asserts that nodes connected via subproperties are also connected via the corresponding superproperties. Note that, the algorithm excludes reflexive rules, i.e, `tp` being sub-property or equivalent to itself. The omission of these entailed triples does not affect the SHACL validation process, as this constitutes redundant information.

This phase then addresses equivalent properties (lines 17–22). The function ϕ_{EP} retrieves the identical (`owl:sameAs`) and equivalent (`owl:equivalentProperty`) properties associated with `tp` from the data graph G, consolidating them within the set denoted as EP. This function handles the transitivity of equivalences to compute the full set of semantically equivalent properties of `tp`. Next, for each equivalent property `sp` to `tp`, the algorithms substitutes `sp` by `tp` in the triples where `sp` occurs, following the rules *eq-rep-p*, *eq-rep-eqp1*, and *eq-rep-eqp2*. The entailed triples are materialized in G. Subsequently, the triples that involve `tp` with `owl:sameAs` and `owl:equivalentProperty` are removed from the data graph to avoid redundant operations (line 22).

Then, this phase enhances the input graph with information concerning the target path and its corresponding inverse property using the rules *prp-inv1* and *prp-inv2* (lines 23–27). Lastly, in the cases where the target path `tp` is a symmetric, transitive, functional, or inverse functional property, the algorithm entails new triples obtained with rules *prp-symp*, *prp-trp*, *prp-fp*, and *prp-ifp* (lines 31–39). Including these entailed triples in the graph G allows Re-SHACL to identify nodes that should also be targeted during validation.

Example 4.3. Consider the data in Fig. 3 and the structures in Example 4.1. Let `fp := dbo:spouse` be the target property we are working with. Since `fp` has no sub-properties and no equivalent properties in G, and the triple `(dbo:spouse, rdf:type, owl:SymmetricProperty)` declares `dbo:spouse` to be a symmetric property, line 28 in Alg. 2 is triggered. According to the triple `(dbr:Marie_Curie, dbo:spouse, dbr:Pierre_Curie)` and the rule *prp-symp*, the statement `(dbr:Pierre_Curie,dbo:spouse, dbr:Marie_Curie)` can be entailed (lines 20-30). Then, the processing for `fp` in Alg. 2 is complete. After reasoning, the validation of the shape `:ss` is performed over `dbr:Pierre_Curie` because the new inferred triple makes it have the target property `dbo:spouse`.

Alg. 2 is executed until no new triples are entailed. The Re-SHACL reasoning engine excludes rules that are not pertinent to the validation process. For example, the rule *scm-sco* (or *rdfs11*) about the transitivity of `rdfs:subClassOf` does not feature in the algorithm as it has been already addressed in the construction of C in Phase 1. Moreover, reasoning concerning general definitions such as `rdfs:Resource` or `rdfs:Datatype` is not covered in Re-SHACL, as it is customary in SHACL to explicitly specify datatypes (e.g., `xsd:date` and `xsd:float`) in the shapes graph. Furthermore, *scm-cls* (class definition), *scp-dp* (datatype property definition), and

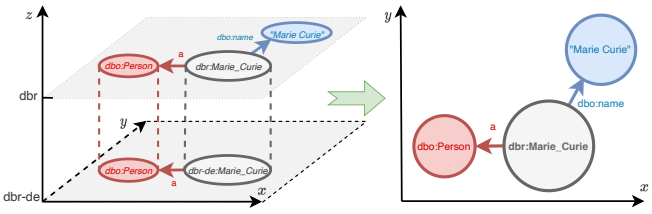


Figure 4: Illustration of the Re-SHACL merging technique

scm-op (object property definition) are also omitted, as they do not entail relevant triples for the validation.

4.2.2 Shape-based node merging. In reasoning, the triples of entities connected via `owl:sameAs` are copied over each entity. This generates a large amount of duplicate information in the graph. To avoid this, Re-SHACL implements a merging approach where semantically equivalent entities are combined into a single, unified entity that encompasses all the triples from its corresponding `owl:sameAs` entities. Fig. 4 illustrates the merging approach with the example in Fig. 1. In the RDF graph, `dbr:Marie_Curie` denotes an identical entity to `dbr-de:Marie_Curie`, albeit having different identifiers. Our method merges these two nodes into a single representation denoted by the unique name "`dbr:Marie_Curie`". In this case, we say that the node `dbr-de:Marie_Curie` has been *subsumed* by `dbr:Marie_Curie`. Fig. 4(c) shows the resulting merged graph, which is more concise (with two triples) in comparison to the graph obtained with reasoning (containing six triples).

To minimize the runtime, Re-SHACL applies this merging strategy only to focus nodes, which are relevant for validation. For each focus node `fn` in F, Re-SHACL checks whether there exists equivalent nodes `sn` defined with the triples $t = (\texttt{fn}, \texttt{owl:sameAs}, \texttt{sn})$ or $t = (\texttt{sn}, \texttt{owl:sameAs}, \texttt{fn})$ in the data graph G, which has been enhanced with implicit statements using targeted reasoning (§ 4.2.1). Subsequently, all instances of `sn` in the data graph are substituted with `fn`, i.e., `sn` is subsumed by `fn`. The triple t can safely be excluded from the merged graph, only when the property `owl:sameAs` does not occur in the target path of the focus node. For example, if a shape for the focus node `dbr:Marie_Curie` states that the entity should have at least one `owl:sameAs` value, the merging process depicted in Fig. 4 would introduce a spurious violation, as the `owl:sameAs` triples are not included in the merged graph. Re-SHACL takes this into account, thus, only removes the triples t from the merged graph when applicable.

Lastly, to ensure the preservation of critical equivalence information throughout the merging process, Re-SHACL updates the dictionary D to capture the equivalence relationships between the merged nodes `sn` and the designated focus node `fn`, as follows:

- if $\texttt{sn} \notin D$, then $D[\texttt{fn}] \leftarrow D[\texttt{fn}] \cup \{\texttt{sn}\}$;
- else, then $D[\texttt{fn}] \leftarrow D[\texttt{fn}] \cup D[\texttt{sn}] \cup \{\texttt{sn}\}$.

The merging process adheres to the node equality semantics in OWL LD and OWL 2 RL, but omits the rule *eq-ref* (reflexivity) to avoid redundant information. The reasoning and merging engine performs multiple iterations of consolidation operations on the data graph until the subgraphs associated with validation achieve complete UNA compliance. Finally, the reasoning and merging engine yields and concise merged graph G' from the data graph G.

4.2.3 Shapes graph re-writing. Re-SHACL rewrites the shapes graph when a target node is subsumed into other focus nodes. To illustrate this, consider `tn` a target node in a shape `s`, and `tn` is subsumed by another focus node during the merging process. In this case, the merged graph no longer retains the original identifier associated with `tn` from the input RDF graph. Consequently, the shape `s` is unable to identify the target node `tn` in the merged graph G'. To avoid such a spurious behavior, a lightweight rewriting of the shapes graph is performed, that substitutes the occurrences of `tn` with the target node's new identifier in the merged graph using the dictionary D. Specifically, if a target node $\texttt{tn} \in N$ undergoes a merging operation with a focus node `f`, it holds that $\texttt{tn} \in D[\texttt{f}]$, and Re-SHACL systematically identifies all the shapes s in the shapes graph that designate `tn` as the target node. Then, the method updates $targNode_s = (targNode_s - \{\texttt{tn}\}) \cup \{\texttt{f}\}$. In practice, this is done by replacing the triple `t=(s,sh:targetNode,tn)` in the shapes graph with the new triple `t'=(s,sh:targetNode,f)`.

Example 4.4. Based on the example in Fig. 3 and initialization in Example 4.1, the target node $\texttt{tn} := \texttt{dbr-de:Marie\_Curie}$ of the shape `:ss` is subsumed into the node $\texttt{f} := \texttt{dbr:Marie\_Curie}$ during the merging process. Then, Re-SHACL will replace the triple `t:=(:ss,sh:targetNode,dbr-de:Marie_Curie)` in $\mathcal{S}$ with the triple `t':=(:ss,sh:targetNode,dbr:Marie_Curie)`. Such rewriting will not affect the validation because `tn` and `f` are semantically the same entity. Finally, N is updated to $N := \{\texttt{dbr:Marie\_Curie}\}$.

4.3 Re-SHACL Theoretical Results

First, we look into the complexity of our approach. The presented reasoning and merging techniques rely on the semantics of OWL LD, which is a subset of the OWL 2 RL fragment. The data complexity of OWL 2 RL is known to be polynomial [32] w.r.t. to the number of assertions in the input data, in this case, G. Furthermore, since the entailment rules used in Re-SHACL to compute the merged graph G' do not introduce new nodes that are not in G, the proposed algorithms are also deterministic. Therefore, from these results established in the literature, we conclude that Re-SHACL also provides the same theoretical guarantees.

Lemma 4.5. *Re-SHACL is deterministic and runs in polynomial time w.r.t.* $|G|$.

Lemma 4.6. $|G'|$ *is polynomial w.r.t.* $|G|$.

Next, we look into the soundness of the lightweight rewriting techniques of Re-SHACL to obtain $\mathcal{S}'$ from the shapes graph $\mathcal{S}$.

Lemma 4.7. $|\mathcal{S}'|$ *is sound.*

Proof. Consider a shape `s`, and a target node entity `tn` which has been subsumed by another node `f` during merging and not targeted by `s`. By contradiction, assume the rewriting of `s` into `s'` is incorrect, i.e., $\texttt{f} \notin targNode_{s'}$. If $\texttt{f} \notin targNode_{s'}$, then (i) $\texttt{f} \notin D$, or (ii) $\texttt{tn} \notin D[\texttt{f}]$. (i) entails that f has not subsumed equivalent target nodes during the merging process. (ii) entails that `tn` was not subsumed by `f`. Both cases contradict our hypothesis. □

5 EXPERIMENTAL STUDY

In the experiments, we investigate the following questions: **(Q1)** What is the impact of Re-SHACL components in isolation? **(Q2)** How do the data graph, shapes graph, and entailment affect the validation performance? **(Q3)** How many and what type of violations can be detected with and without entailment? **(Q4)** What is the runtime efficiency of state-of-the-art validators combined with Re-SHACL? Resources to reproduce our study are available online.

5.1 Experimental Setup

Data Graphs and Shapes Graphs. We use nine synthetic datasets from LUBM [19] and six real-world subsets obtained from DBPedia [28] and Wikidata [43]. For LUBM, we use the graphs generated by Figuera et al. [15]. For DBpedia and Wikidata, we randomly created subsets with different numbers of entities per targeted class. For the shapes graphs, Figuera et al. [15] provided three constraint datasets for LUBM denoted Schema1, 2, and 3. The DBpedia and Wikidata shapes graphs were obtained from the constraints mined by Rabbani et al. [38]; from this graph, we selected shapes without references to other shapes, as this complicates the analysis of the detected violations. Altogether, we have 33 configurations (27 LUBM, 3 DBpedia, 3 Wikidata) of data-shape graphs in our experiments. Table 1 summarizes the datasets' descriptions.

Studied Approaches. We use pySHACL [40], Trav-SHACL [14], and SHACL2SPARQL [6, 7]. pySHACL is an open-source library that supports RDFS and OWL RL reasoning during validation. This enables a comparison with Re-SHACL. SHACL2SPARQL and Trav-SHACL are also open source and used here as off-the-shelf validators to show how Re-SHACL can enable scalable validators to incorporate reasoning. We used the best configurations of Trav-SHACL and SHACL2SPARQL as reported by the authors.

Implementation Details. Re-SHACL, pySHACL, and Trav-SHACL are implemented in Python3 (3.8). SHACL2SPARQL [8] is implemented in Java (openjdk 11.0.16). For pySHACL, the data and shapes graph are loaded into main memory, followed by a warm up stage, and then the execution of the approaches starts. Trav-SHACL and SHACL2SPARQL run over a Virtuoso SPARQL endpoint (Open Source v.7.2). All experiments have been conducted on a dual socket AMD EPYC 7713 server with 128 cores and 1TB of main memory.

Metrics. Each approach is executed three times[1] over the 31 configurations of data-shapes graph. We report on: *(i) Graph size:* Number of triples in a graph. *(ii) Runtime:* Elapsed time (in seconds) since the approach is invoked until it finishes. *(iii) Number of Violations:* Overall number of violations detected with the approach.

5.2 Impact of Re-SHACL Components on Performance: Ablation Study

We conduct an ablation study to determine the effects of the Re-SHACL techniques in isolation. We measure the effects of these techniques during graph materialization and validation. A reduction in runtime in any of these tasks directly improves the overall performance of the knowledge graph validation pipeline. The studied Re-SHACL techniques are:

[1]Some configurations could only be run once due to the long runtimes.

Table 1: Dataset descriptions

Data Graph	Triples	Subj.	Pred.	Obj.	SameAs
LUBM (SKG)	1,001,716	163,701	18	122,989	0
LUBM (MKG)	4,258,329	693,909	18	516,901	0
LUBM (LKG)	34,106,115	5,549,100	18	4,125,586	0
DBpedia (DB50)	534,696	134,303	2,888	140,855	26,624
DBpedia (DB100)	912,108	208,836	3,463	229,587	47,325
DBpedia (DB1000)	3,382,528	490,612	5,713	767,604	169,937
Wikidata (WDS)	4,974,498	4,159,871	709	288,610	106
Wikidata (WDM)	13,268,141	5,336,919	12,769	2,428,390	7,260
Wikidata (WDL)	29,101,498	7,339,402	20,380	6,508,412	21,601

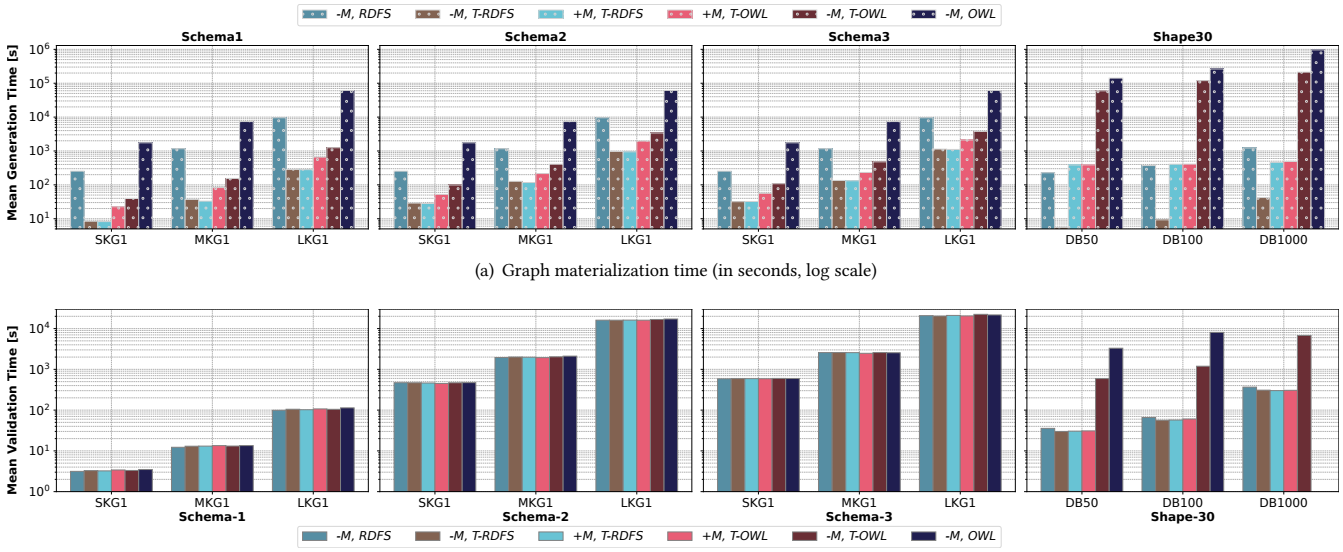
Shapes Graph	Node Shapes	Property Shapes	Targeted Classes	Targeted Properties	Shape References
LUBM (Schema1)	3	12	3	10	Yes
LUBM (Schema2)	7	24	7	16	Yes
LUBM (Schema3)	14	79	14	17	Yes
DBpedia (Shape30)	30	611	30	255	No
Wikidata (Shape100)	100	53,087	100	7,719	No

- Merging techniques: +M indicates the merging of *same as* nodes (and the resp. rewriting), -M indicates no merging.
- Targeted reasoning: T-*E* applies reasoning on focus nodes using the entailment regime *E*, otherwise it computes the graph closure under *E*.
- Entailment regime: RDFS or OWL entailment regime.

The ablation study comprises six configurations. The configurations (-M, RDFS) and (-M, OWL) correspond to baselines, i.e., the approach computes the (full) closure of the graphs for the given entailment regime, without merging semantically equivalent nodes. The remaining four configurations allow for turning on/off the merging and targeted reasoning techniques of Re-SHACL. For the validation task, we use PySHACL.

In this study, we selected the three DBpedia datasets (DB50, DB100, DB1000) and three LUBM graphs (SKG1, MKG1, LKG) to have real-world and synthetic graphs of varying sizes.

Impact of merging techniques. To study the effect of activating the Re-SHACL merging techniques, we compare (+M, T-RDFS) with (-M, T-RDFS), and (+M, T-OWL) with (-M, T-OWL) in Fig. 5. Since LUBM has no owl:sameAs links, the Re-SHACL merging techniques with RDFS have no effect. However, during LUBM materialization, (+M, T-OWL) is, on average, 1.87x faster than (-M, T-OWL), as the merging techniques cope with the owl:sameAs reflexivity, thus avoiding the generation of redundant triples. All these observations apply to the three shapes graphs (Schema-1, 2, 3) in LUBM. Since DBpedia has owl:sameAs links (cf. Table 1), the activation of the merging techniques impacts the materialization and validation runtimes. First, comparing (+M, T-RDFS) with (-M, T-RDFS), we observe that node merging introduces an overhead, as it requires consolidating thousands of entities in DBpedia, as shown in Table 2. Yet, the configurations (+M, T-OWL) and (-M, T-OWL) clearly show that the merging techniques reduce the materialization and validation runtimes up to three orders of magnitude when performing targeted OWL reasoning. These results are supported when examining the size of the materialized graphs in Table 3. In the extreme case of DB1000, the merging techniques with targeted reasoning can reduce the graph size from 116M to 4.12M triples.



(a) Graph materialization time (in seconds, log scale)

(b) Graph validation time (in seconds, log scale). (-M, OWL) timed out for DB1000

Figure 5: Results for the ablation study. The notation is as follows:+M=merging; -M=no merging; T-=targeted reasoning.

Table 2: Re-SHACL merged nodes in KGs with *same as* triples

Dataset	DB50	DB100	DB1000
Merged entities	20,426	36,725	129,049
Uniform representations created during merging	1,260	2,307	9,137

Table 3: Size of materialized graphs in the ablation study. For targeted reasoning (T-) over LUBM, we report the average graph size across all shapes graphs

		LUBM			DBpedia		
		SKG1	MKG1	LKG1	DB50	DB100	DB1000
Baseline	(-M, RDFS)	1.5M	6.4M	51.0M	1.2M	2.0M	6.6M
	(-M, T-RDFS)	1.1M	4.8M	38.6M	0.69M	1.15M	4.15M
	(+M, T-RDFS)	1.1M	4.8M	38.6M	0.68M	1.12M	4.07M
Baseline	(-M, OWL)	2.2M	9.5M	76.0M	32.0M	57.6M	175M
	(-M, T-OWL)	1.3M	5.7M	45.6M	21.0M	38.3M	116M
	(+M, T-OWL)	1.3M	5.4M	43.2M	0.69M	1.15M	4.12M

Impact of targeted reasoning. Next, we compare the effects of targeted reasoning without the merging techniques. In LUBM, Fig. 5(a) shows that the materialization time of (-M, T-RDFS) is 10x faster than (-M, RDFS), and (-M, T-OWL) is up to 100x faster than (-M, OWL). These results confirm that applying (targeted) reasoning to the focus nodes and not the entire graph effectively reduces the materialization time. Still, since the size of the LUBM materialized graphs (cf. Table 3) with and without targeted reasoning is in the same order of magnitude, the validation runtimes are almost the same across all configurations. In DBpedia, the targeted reasoning greatly reduces the materialization time, as shown in the results for (-M, T-RDFS) and (-M, RDFS). In the case of OWL, the targeted reasoning reduces the materialization times by half in DB50 and DB100, and by a factor of 0.8 in DB1000 (cf. Fig 5(a), in log scale). The validation runtimes exhibit similar trends, where in comparison to (-M, OWL), (-M, T-OWL) reduces the runtime by factors of 0.82 (DB50) and 0.85 (DB100), while (-M, OWL) timed out after 14 days. The DBpedia runtimes are consistent with the size of the materialized graphs (cf. Table 3), i.e., the size of the graphs with T-OWL is considerably smaller than their OWL counterpart.

Impact of entailment regimes. In this study, OWL reasoning is orders of magnitude more expensive than RDFS. This can be observed in our results when comparing the configurations (-M, RDFS) and (-M, OWL). In the case of DBpedia, while the RDFS closure was computed in 20 minutes, the OWL closure took over 268 hours (almost 12 days), i.e., OWL reasoning is impractical. Yet, our Re-SHACL techniques can reduce the OWL reasoning runtime significantly, making it comparable to the RDFS reasoning runtime. In LUBM, since the merging techniques have no effects due to the absence of owl:sameAs links, we observe that the targeted RDFS reasoning is between 1.7x and 3x faster than targeted OWL reasoning across all datasets and schemas. Yet, since DBpedia contains owl:sameAs, the merging techniques take a large portion of the materialization time, making the performance of the configurations (+M, T-RDFS) and (+M, T-OWL) nearly identical. Although RDFS does not handle the semantics of owl:sameAs, activating the merging techniques allows for consolidating equivalent entities, thus avoiding spurious validations caused by having partial information attached to only one of the entity representations. The violations detected with the different approaches will be discussed in Sect. 5.4.

The ablation study demonstrates that our merging and targeted reasoning techniques reduce by orders of magnitude the materialization runtime compared to standard reasoning **(Q1)**.

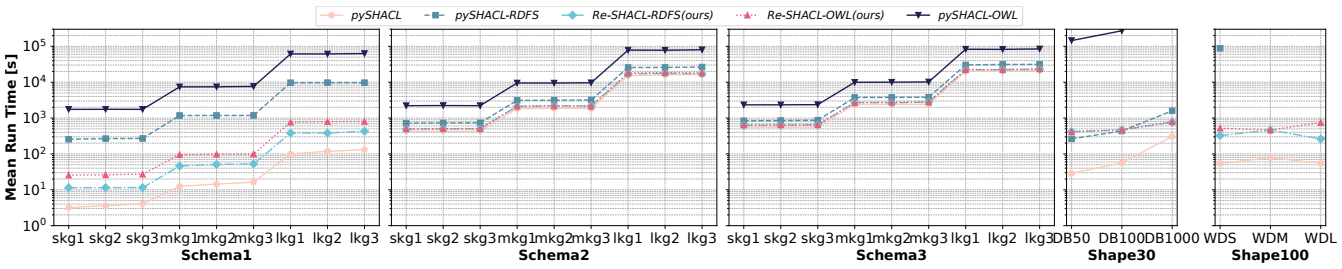


Figure 6: Validation pipeline runtime (in seconds) on all datasets. Approaches not shown due to timeout

5.3 Validation Pipeline Runtime with and without Reasoning

For approaches with entailment, the validation pipeline includes reasoning and the actual graph validation time. We report on the proposed Re-SHACL configurations, i.e., (+M, T-RDFS) as Re-SHACL RDFS and (+M, T-OWL) as Re-SHACL OWL. For validation, Re-SHACL uses PySHACL. As baseline, we use PySHACL (with no entailment), and PySHACL-RDFS and PySHACL OWL. Fig. 6 shows the entire validation process runtime over all datasets.

Impact of the data graph size. In LUBM, the performance of all the approaches is, on average, slowed down by one order of magnitude when going from SKGs (1M triples) to MKGs (4.5M triples), and then to LKGs (34M triples). This effect is observed for all configurations of Re-SHACL and PySHACL. In the case of DBpedia and Wikidata, increasing the size of the KGs negatively affects the performance of PySHACL. This becomes evident when comparing DB50 (0.5M triples) and DB100 (3.4M triples), where PySHACL-OWL takes 12 days to compute the closure and did not finish the validation before 7 days. In DBpedia and Wikidata, the performance of Re-SHACL (both RDFS and OWL) is not significantly impacted by the graph size and remains in the same order of magnitude.

Impact of the data graph expressivity. Interestingly, PySHACL-OWL in LUBM MKGs (with ~4M triples) is orders of magnitude faster than in DBpedia graphs (with ~0.5 and 1M triples). The difference in runtime is due to the fact that LUBM does not contain `owl:sameAs` triples, unlike DBpedia. Recall that `owl:sameAs` introduces a large number of irrelevant triples due to its reflexivity, symmetry, and transitivity. This shows that the performance of traditional reasoning approaches is not only affected by the graph size, but the semantics of terms in the data graph and the entailment regime. In comparison, since Re-SHACL implements merging and targeted reasoning, the increase in expressivity in the data graph does not have a significant impact on its performance. Contrary, the presence of owl:sameAs links is exploited by Re-SHACL, thus achieving lower runtimes in DBpedia than in LUBM MKG.

Impact of the shapes graph size and structure. Our results show that the shapes directly impact the approaches' performance. In LUBM, the performance of all approaches becomes similar (except for PySHACL-OWL) for Schema2 and Schema3, as these shapes graphs have more shapes and targeted classes than Schema1. Also, in LUBM Schema1, Re-SHACL is faster than for other shapes graphs, regardless of the size of the data graph. This is because Re-SHACL takes the shapes graph into account during the targeted reasoning, and Schema1 has a simpler structure than the other shapes graphs. When comparing datasets of similar sizes, e.g., LUBM MKG with DBpedia and Wikidata, we observe that the approaches' performance differs significantly. The validation of LUBM MKG Schema2 and Schema3 is around 10x slower than DBpedia (except for PySHACL-OWL), because these schemas contain shape references. Therefore, besides the size, the shapes graph structure also affects the runtime.

Entailement vs. no reasoning. Enabling reasoning introduces an overhead compared to just performing validation on the input data graph. In graphs where the validation process is simple (e.g., LUBM Schema1), Re-SHACL outperforms the PySHACL counterparts. In the other graphs, PySHACL-OWL is always the slowest and even timed out for DB1000 and Wikidata. In contrast, Re-SHACL can be executed over all tested datasets in a reasonable amount of time and is able to scale to large datasets up to 34M triples.

In summary, all the aspects studied in the experiments affect the validation performance. Yet, in contrast to the baselines, Re-SHACL can efficiently perform reasoning and cope with the effects of larger and more expressive graphs **(Q2)**.

5.4 Detected Violations

In this section, first, we discuss the number of violations detected with the studied approaches (cf. Table 4). Next, we analyze the differences between the validation reports produced by the approaches using a selected dataset (cf. Fig 7).

Impact of entailment regimes in the number of violations. Table 4 shows the number of violations detected. In LUBM, the constraints in Schema1 target nodes that do not contain further ontological definitions and, in consequence, all the approaches produce the same number of violations. For Schema2 and Schema3, Re-SHACL tends to detect more violations than pySHACL (simple) but less than its counterparts. Also in LUBM, Re-SHACL and pySHACL-OWL produce the same number of violations, which indicates that the Re-SHACL targeted reasoning correctly produces all relevant triples for validation, but in a faster and more compact way. In the case of DBPedia, we observe that all the approaches report different numbers of violations. This is because DBpedia contains RDFS and OWL constructs, thus, detecting difderent violations results when enabling a specific entailment regime. In DB100, pySHACL-OWL reports over 1M violations while Re-SHACL only 54K as it avoids duplicate violations due to its merging techniques.

Erroneous violations filtered with entailment. Fig. 7(a) and (b) show that PySHACL OWL and Re-SHACL OWL effectively remove 2611 and 148 spurious violations, respectively. Although PySHACL OWL removes more violations than Re-SHACL, this is not necessarily a

Table 4: Numbers of reported violations. Using simple validation (no entailment) as a reference, highlighted cells indicate differences in the number of violations. The darker the cell, the higher the difference w.r.t. simple validation

Dataset		PySHACL	PySHACL RDFS	Re-SHACL RDFS	PySHACL OWL	Re-SHACL OWL
Schema1	SKG1	2,136	2,136	2,136	2,136	2,136
	SKG2	2,938	2,938	2,938	2,938	2,938
	SKG3	3,703	3,703	3,703	3,703	3,703
	MKG1	5,836	5,836	5,836	5,836	5,836
	MKG2	9,190	9,190	9,190	9,190	9,190
	MKG3	12,424	12,424	12,424	12,424	12,424
	LKG1	40,000	40,000	40,000	40,000	40,000
	LKG2	66,173	66,173	66,173	66,173	66,173
	LKG3	92,289	92,289	92,289	92,289	92,289
Schema2	SKG1	197,784	197,784	197,784	202,250	202,250
	SKG2	198,586	198,586	198,586	203,052	203,052
	SKG3	199,351	199,351	199,351	203,817	203,817
	MKG1	834,143	834,143	834,143	853,343	853,343
	MKG2	837,497	837,497	837,497	856,697	856,697
	MKG3	840,731	840,731	840,731	859,931	859,931
	LKG1	6,419,120	6,694,924	6,694,924	6,848,446	6,848,446
	LKG2	6,445,293	6,721,097	6,721,097	6,874,619	6,874,619
	LKG3	6,471,409	6,747,213	6,747,213	6,900,735	6,900,735
Schema3	SKG1	288,610	288,610	288,610	293,076	293,076
	SKG2	297,536	297,536	297,536	302,002	302,002
	SKG3	306,472	306,472	306,472	310,938	310,938
	MKG1	1,224,043	1,224,043	1,224,043	1,243,243	1,243,243
	MKG2	1,262,285	1,262,285	1,262,285	1,281,485	1,281,485
	MKG3	1,300,837	1,300,837	1,300,837	1,320,037	1,320,037
	LKG1	9,534,325	9,810,129	9,810,129	9,963,651	9,963,651
	LKG2	9,840,368	10,116,172	10,116,172	10,269,694	10,269,694
	LKG3	10,146,652	10,422,456	10,422,456	10,575,978	10,575,978
Shape30	DB50	27,392	31,421	28,547	871,381	30,019
	DB100	49,537	56,861	51,339	1,502,071	54,030
	DB1000	200,730	227,303	201,638	–	213,800
Shape100	WDS	0	0	0	–	0
	WDM	0	0	0	–	0
	WDL	0	0	0	–	0

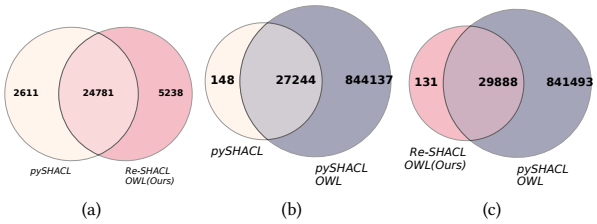


Figure 7: Analysis of violation reports for DB50

desirable behavior. Re-SHACL minimizes the number of detected violations due to the merging techniques, which avoid reporting duplicate violations for entities with owl:sameAs.

Novel violations detected with entailment. Fig. 7(a) and (b) show these cases on the right circles. In comparison to PySHACL (simple), Re-SHACL OWL produces and PySHACL OWL produce 5, 238 and 844, 137 novel violations. Yet, producing more violations does not mean better behavior, as some of these violations are duplicates generated by the nUNA, as discussed below.

Duplicate violations due to non-Unique Named Assumption (nUNA). Enabling reasoning can amplify the negative impact of nUNA – introduced with the usage of `owl:sameAs`– in SHACL validation. This can be seen in Fig. 7 (right), where OWL reasoning reports

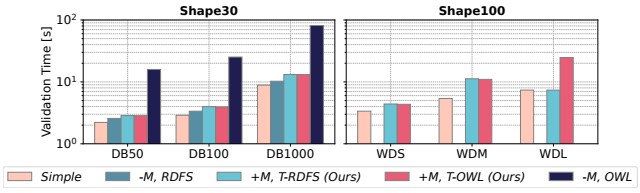


Figure 8: SHACL2SPARQL runtime over real-world datasets

841, 493 more violations than Re-SHACL OWL. After inspecting the validation reports, we confirm that these massive number of violations are due to semantic duplicates: a violation that occurs in an entity is reported multiple times, i.e., one for each identifier of the semantically equivalent terms. In contrast, Re-SHACL reports the violation once as it supports OWL entailment with entity merging. Still, with Re-SHACL it is possible to allocate all the semantically equivalent entities affected by a violation using the dictionary D.

Violations involving reflexive properties and cardinality constraints. During reasoning, reflexive triples `(e,owl:sameAs,e)` produce spurious validation reports. Such entailed triples can easily conflict with the `sh:minCount` and `sh:maxCount` cardinality constraints. An inspection of the validation reports shows that the 131 violations captured by Re-SHACL OWL but not recognized by pySHACL-OWL in DB50 (cf. Fig. 7) are due to this issue.

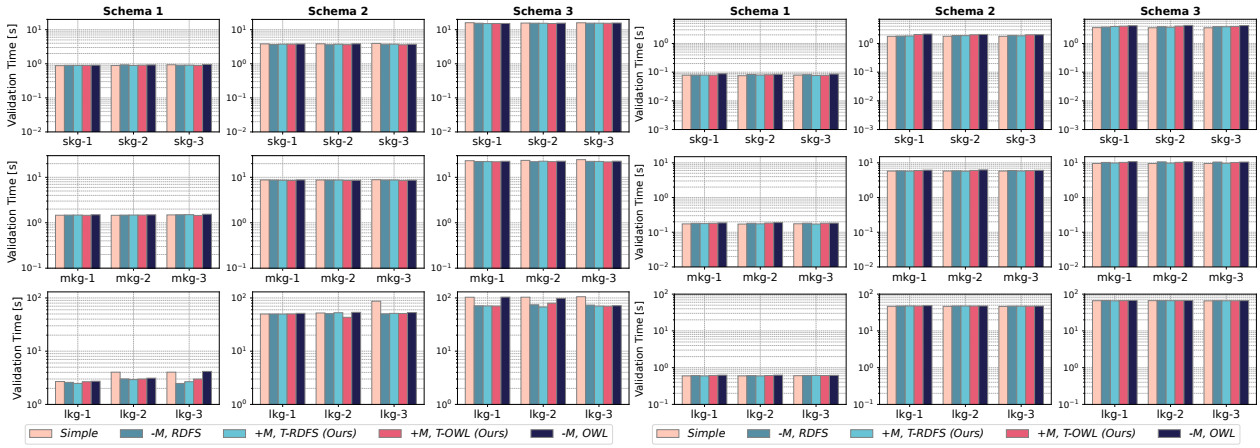
In summary, reasoning allows for detecting novel violations, while pruning false violations when implicit knowledge is ignored. Furthermore, the Re-SHACL merging techniques remove redundant duplicates from semantic equivalences in the KG, thus producing fewer violations than the RDFS and OWL counterparts **(Q3)**.

5.5 Runtime of State-of-the-art Validators

We now investigate the performance of SHACL2SPARQL and Trav-SHACL when using the original data graph (Simple), the graphs generated with closures, and our Re-SHACL.

In the case of DBpedia, we were only able to run SHACL2SPARQL as Trav-SHACL (by design) does not support all types of constraints in the Shapes30 and Shapes100 graphs. Fig. 8 shows the runtime for SHACL2SPARQL on DBpedia and Wikidata. In DBpedia, the behavior of Re-SHACL is very similar to the baselines with no entailment and RDFS. Yet, in comparison to the OWL closure (-M, OWL), the runtime over the Re-SHACL graph is significantly faster. This is due to the size of the resulting graphs after reasoning (cf. Table 3), where the OWL closure for DB1000 contains 175M triple, while Re-SHACL OWL produces 4.12M. In this case, the size of the graphs greatly impact the performance of SHACL2SPARQL. For Wikidata, SHACL2SPARQL could not execute the baselines due to timeouts (after 10^{12} seconds) raised by the endpoint. Yet, all the configurations of Re-SHACL could be run with SHACL2SPARQL. Only Re-SHACL OWL is impacted by the size of the graph in WDL.

Next, we look into the performance of the validators for LUBM reported in Fig. 9. For SHACL2SPARQL (cf. Fig. 9(a)), in the majority of the cases, there are no significant differences in the SHACL2SPARQL runtime over the different graphs. Lastly, the results for Trav-SHACL in Fig. 9 show that, on average, the fastest performance is achieved for the original data graph (Simple). Still, we observe that in most of the cases, Trav-SHACL is slightly faster (on average)



(a) SHACL2SPARQL runtime over LUBM

(b) Trav-SHACL runtime over LUBM

Figure 9: Validation time of state-of-the-art validators over synthetic datasets

using Re-SHACL than with the RDFS and OWL counterparts, although there are no significant differences. In summary, for LUBM, the obtained graphs are relatively small (cf. Table 3), and since they are loaded into a SPARQL endpoint (which implements advanced querying techniques) and are processed by validators that implement more sophisticated validation techniques, the effects of the graph size on performance is minimal. Still, let us keep in mind that these runtimes do not account for the reasoning time and that computing the OWL closure takes one or even several orders of magnitude longer than using Re-SHACL.

From this experiment, we conclude that state-of-the-art validators can greatly benefit from Re-SHACL to seamlessly support SHACL validation with entailment efficiently **(Q4)**.

6 DISCUSSION

In this section, we discuss some limitations and open problems of including entailment regimes in SHACL validation.

Lack of semantics of SHACL validation under entailment regimes beyond OWL 2 QL. Recent works [1] have proposed a semantics for SHACL with OWL 2 QL entailment, which does not account for `owl:sameAs` or more expressive constructs in OWL 2 RL. Still, a formal semantics of SHACL validation under entailment that accounts for the semantics of `owl:sameAs` is imperative to probe the theoretical correctness of approaches.

Tradeoff between expressivity and runtime. Our experiments show that enabling entailment during SHACL validations always introduces an overhead. However, if the semantics of terms in the knowledge graph is disregarded, SHACL validation without entailment may compromise the accuracy of the results. Therefore, users must carefully weigh the benefits and drawbacks of using entailment and select the appropriate expressivity that meets the specific requirements of their applications.

Scalability to massive graphs. Existing strategies for incorporating entailment in SHACL validation are prone to generate massive data graphs or rewritings. This hinders the applicability of these techniques to datasets with billions of triples. A possible solution is to partition the graph, yet this may yield incomplete or incorrect violations. Therefore, techniques that account for a balance between scalability and correctness must be devised.

7 CONCLUSION AND FUTURE WORK

This work presents Re-SHACL, a solution for efficient SHACL validation with reasoning. Unlike existing approaches, Re-SHACL leverages a shape-based reasoning and merging approach to support entailment during SHACL validation efficiently. Specifically, Re-SHACL first analyzes the shapes graph to obtain relevant information and then implements reasoning, merging, and rewriting strategies to minimize unnecessary inferred triples in the data graph. By doing so, Re-SHACL can significantly reduce the workload in the inference process and mitigate undesired behaviors during validation introduced by the entailment regime. As a result, Re-SHACL enables efficient SHACL validation with entailment, even in state-of-the-art validators, while providing accurate results.

Application of our contributions to other models. The Re-SHACL merging techniques can be effectively applied to other models, particularly in data fusion scenarios where multiple sources do not adhere to the UNA. Also, our targeted reasoning could be adapted to other semi-structured models, e.g., XML and JSON-LD, if the sources include logical definitions, e.g., concept hierarchies.

Future work. Besides the open problems discussed in Sect. 6, future work also includes extending Re-SHACL to support OWL 2 RL and advanced features of SHACL, enabling more complex validations. Another line of research is to devise techniques to perform validations over data graph partitions. This will allow entailment-based approaches to scale up to very large datasets.

ACKNOWLEDGMENTS

The authors thank Thomas Neumann, Sebastian Rudolph, Katja Hose, and the reviewers for their valuable input. This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1608 - 501798263, and SFB 1625 - 506711657, subproject A06.

REFERENCES

[1] Shqiponja Ahmetaj, Magdalena Ortiz, Anouk Oudshoorn, and Mantas Šimkus. 2023. Reconciling SHACL and ontologies: semantics and validation via rewriting. In *ECAI 2023*. IOS Press, Kraków, Poland, 27–35.

[2] Dan Brickley and R. V. Guha. 2014. *RDF Schema 1.1*. W3C. Retrieved 10 July 2024 from https://www.w3.org/TR/rdf-schema/

[3] Antoon Bronselaer and Maribel Acosta. 2023. Parker: Data fusion through consistent repairs using edit rules under partial keys. *Inf. Fusion* 100 (2023), 101942.

[4] Peter Buneman, Wenfei Fan, Jerome Simeon, and Scott Weinstein. 2001. Constraints for semistructured data and XML. *ACM Sigmod Record* 30, 1 (2001), 47–54.

[5] Xu Chu, Ihab F Ilyas, and Paolo Papotti. 2013. Discovering denial constraints. *Proceedings of the VLDB Endowment* 6, 13 (2013), 1498–1509.

[6] Julien Corman, Fernando Florenzano, Juan L. Reutter, and Ognjen Savkovic. 2019. SHACL2SPARQL: Validating a SPARQL Endpoint against Recursive SHACL Constraints.. In *Proceedings of the ISWC 2019 Satellite Tracks (Posters & Demonstrations, Industry, and Outrageous Ideas) (CEUR Workshop Proceedings)*, Vol. 2456. CEUR-WS.org, Auckland, New Zealand, 165–168.

[7] Julien Corman, Fernando Florenzano, Juan L. Reutter, and Ognjen Savkovic. 2019. Validating Shacl Constraints over a Sparql Endpoint. In *The Semantic Web, International Semantic Web Conference (Lecture Notes in Computer Science)*, Vol. 11778. Springer, Auckland, New Zealand, 145–163.

[8] Julien Corman, Fernando Florenzano, Juan L Reutter, and Ognjen Savkovic. 2021. *SHACL2SPARQL*. Retrieved 10 July 2024 from https://github.com/rdfshapes/shacl-sparql

[9] Julien Corman, Juan L. Reutter, and Ognjen Savkovic. 2018. Semantics and Validation of Recursive SHACL. In *The Semantic Web, International Semantic Web Conference (Lecture Notes in Computer Science)*, Vol. 11136. Springer, Monterey, CA, USA, 318–336.

[10] Wenfei Fan and Floris Geerts. 2012. *Foundations of data quality management*. Morgan & Claypool Publishers.

[11] Wenfei Fan, Floris Geerts, Xibei Jia, and Anastasios Kementsietsidis. 2008. Conditional functional dependencies for capturing data inconsistencies. *ACM Transactions on Database Systems (TODS)* 33, 2 (2008), 1–48.

[12] Wenfei Fan and Jérôme Siméon. 2000. Integrity constraints for XML. In *Proceedings of the nineteenth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*. ACM, Dallas, Texas, USA, 23–34.

[13] Ivan P Fellegi and David Holt. 1976. A systematic approach to automatic edit and imputation. *Journal of the American Statistical association* 71, 353 (1976), 17–35.

[14] Mónica Figuera, Philipp D. Rohde, and Maria-Esther Vidal. 2021. Trav-SHACL: Efficiently Validating Networks of SHACL Constraints. In *WWW: The Web Conference*. ACM / IW3C2, Ljubljana, Slovenia, 3337–3348.

[15] Mónica Figuera, Philipp D. Rohde, and Maria-Esther Vidal. 2021. *Trav-SHACL: Benchmarks, Experimental Settings, and Evaluation*. Technische Informationsbibliothek. Retrieved 10 July 2024 from https://doi.org/10.25835/0035739

[16] Birte Glimm. 2011. Using SPARQL with RDFS and OWL Entailment. In *Reasoning Web. Semantic Technologies for the Web of Data (Lecture Notes in Computer Science)*, Vol. 6848. Springer, Galway, Ireland, 137–201.

[17] Birte Glimm, Aidan Hogan, Markus Krötzsch, and Axel Polleres. 2012. OWL: Yet to arrive on the Web of Data?. In *WWW (CEUR Workshop Proceedings)*, Vol. 937. CEUR-WS.org, Lyon, France.

[18] Georg Gottlob, Giorgio Orsi, and Andreas Pieris. 2011. Ontological queries: Rewriting and optimization. In *Proceedings of the International Conference on Data Engineering, ICDE*. IEEE Computer Society, Hannover, Germany, 2–13.

[19] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. 2005. LUBM: A benchmark for OWL knowledge base systems. *Journal of Web Semantics* 3, 2-3 (2005), 158–182.

[20] Thomas Hartmann. 2016. *Validation Framework for RDF-based Constraint Languages*. Ph.D. Dissertation. Karlsruhe Institute of Technology, Germany.

[21] Patrick J. Hayes and Peter F. Patel-Schneider. 2014. *Patterns of RDFS entailment*. W3C. Retrieved 10 July 2024 from https://www.w3.org/TR/rdf11-mt//#patterns-of-rdfs-entailment-informative

[22] Patrick J. Hayes and Peter F. Patel-Schneider. 2014. *RDF 1.1 Semantics*. W3C. Retrieved 10 July 2024 from https://www.w3.org/TR/rdf11-mt/

[23] Pascal Hitzler, Markus Krötzsch, Bijan Parsia, Peter F. Patel-Schneider, and Sebastian Rudolph. 2012. *OWL 2 Web Ontology Language Primer*. W3C. Retrieved 10 July 2024 from https://www.w3.org/TR/owl2-primer/

[24] Daniel Keuchel, Nicolai Spicher, Ju Wang, Michael Völcker, Yang Gong, and Thomas M. Deserno. 2021. SHACL-Based Report Quality Evaluation for Health IT-Induced Medication Errors. In *MEDINFO World Congress on Medical and Health Informatics (Studies in Health Technology and Informatics)*, Vol. 290. IOS Press, 414–418.

[25] Holger Knublauch and Dimitris Kontokostas. 2017. *Shapes Constraint Language (SHACL)*. W3C. Retrieved 10 July 2024 from https://www.w3.org/TR/shacl-js/

[26] Solmaz Kolahi and Laks VS Lakshmanan. 2009. On approximating optimum repairs for functional dependency violations. In *Proceedings of the 12th International Conference on Database Theory*, Vol. 361. ACM, St. Petersburg, Russia, 53–62.

[27] Flip Korn, Barna Saha, Divesh Srivastava, and Shanshan Ying. 2013. On repairing structural problems in semi-structured data. *Proceedings of the VLDB Endowment* 6, 9 (2013), 601–612.

[28] Jens Lehmann, Robert Isele, Max Jakob, Anja Jentzsch, Dimitris Kontokostas, Pablo N. Mendes, Sebastian Hellmann, Mohamed Morsey, Patrick van Kleef, Sören Auer, and Christian Bizer. 2015. DBpedia - A large-scale, multilingual knowledge base extracted from Wikipedia. *Semantic Web* 6, 2 (2015), 167–195.

[29] Ester Livshits, Benny Kimelfeld, and Sudeepa Roy. 2020. Computing optimal repairs for functional dependencies. *ACM Transactions on Database Systems (TODS)* 45, 1 (2020), 1–46.

[30] Johannes Mäkelburg, Christian John, and Maribel Acosta. 2024. Automation of Electronic Invoice Validation Using Knowledge Graph Technologies. In *The Semantic Web: International Conference, ESWC 2024 (Lecture Notes in Computer Science)*, Vol. 14664. Springer, Hersonissos, Crete, Greece, 253–269.

[31] Ben De Meester, Pieter Heyvaert, Dörthe Arndt, Anastasia Dimou, and Ruben Verborgh. 2021. RDF graph validation using rule-based reasoning. *Semantic Web* 12, 1 (2021), 117–142.

[32] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. 2012. *OWL 2 Profiles: Computational Properties*. W3C. Retrieved 10 July 2024 from https://www.w3.org/TR/owl2-profiles/#Computational_Properties

[33] Boris Motik, Bernardo Cuenca Grau, Ian Horrocks, Zhe Wu, Achille Fokoue, and Carsten Lutz. 2012. *OWL 2 Web Ontology Language Profiles*. W3C. Retrieved 10 July 2024 from https://www.w3.org/TR/owl2-profiles/

[34] Paolo Pareti and George Konstantinidis. 2021. A Review of SHACL: From Data Validation to Schema Reasoning for RDF Graphs. In *Reasoning Web. Declarative Artificial Intelligence (Lecture Notes in Computer Science)*, Vol. 13100. Springer, Leuven, Belgium, 115–144.

[35] Paolo Pareti, George Konstantinidis, Timothy J. Norman, and Murat Sensoy. 2019. SHACL Constraints with Inference Rules. In *The Semantic Web - ISWC International Semantic Web Conference (Lecture Notes in Computer Science)*, Vol. 11778. Springer, Auckland, New Zealand, 539–557.

[36] Peter F. Patel-Schneider, Patrick Hayes, and Ian Horrocks. 2004. *OWL Web Ontology Language Semantics and Abstract Syntax*. W3C. Retrieved 10 July 2024 from https://www.w3.org/TR/2004/REC-owl-semantics-20040210/

[37] Abdulhakim Ali Qahtan, Nan Tang, Mourad Ouzzani, Yang Cao, and Michael Stonebraker. 2020. Pattern Functional Dependencies for Data Cleaning. *Proceedings of the VLDB Endowment* 13, 5 (2020), 684–697.

[38] Kashif Rabbani, Matteo Lissandrini, and Katja Hose. 2023. Extraction of Validating Shapes from very large Knowledge Graphs. *Proceedings of the VLDB Endowment*. 16, 5 (2023), 1023–1032.

[39] Ognjen Savković, Evgeny Kharlamov, and Steffen Lamparter. 2019. Validation of SHACL constraints over KGs with OWL 2 QL ontologies via rewriting. In *The Semantic Web - International Conference, ESWC (Lecture Notes in Computer Science)*, Vol. 11503. Springer, Portorož, Slovenia, 314–329.

[40] Ashley Sommer, Nicholas Car, and Jonathan Yu. 2018. *pySHACL*. Retrieved 10 July 2024 from https://pypi.org/project/pyshacl/0.9.5/

[41] Sander Stolk and Kris McGlinn. 2020. Validation of IfcOWL datasets using SHACL. In *Proceedings of the Linked Data in Architecture and Construction Workshop (CEUR Workshop Proceedings)*, Vol. 2636. CEUR-WS.org, Dublin, Ireland, 91–104.

[42] Lars Vogt. 2023. Extending FAIR to FAIREr: Cognitive Interoperability and the Human Explorability of Data and Metadata. *CoRR* abs/2301.04202 (2023).

[43] Denny Vrandecic and Markus Krötzsch. 2014. Wikidata: a free collaborative knowledgebase. *Commun. ACM* 57, 10 (2014), 78–85.

[44] Hong Yao and Howard J Hamilton. 2008. Mining functional dependencies from data. *Data Mining and Knowledge Discovery* 16 (2008), 197–219.