KRDB Research Centre Technical Report:

SHACL Constraint Validation over Ontology-enhanced KGs via Rewriting
[Extended Version]

Ognjen Savković¹, Evgeny Kharlamov², Steffen Lamparter ³

Affiliation
1: Free University of Bozen-Bolzano, Bolzano, Italy
2: University of Oslo, Oslo, Norway
3: Siemens CT, Siemens AG, Munich, Germany

Corresponding author
Ognjen Savković: ognjen.savkovic@unibz.it

Keywords
SHACL, Ontologies RDF Graph Validation, Graph Constraints, RDF, DL-Lite

Number
KRDB18-03

Date
December, 2018

URL
http://www.inf.unibz.it/krdb/
Acknowledgments

This work has been partially funded by the FUB projects QUEST and OBATS.
SHACL Constraint Validation over 
Ontology-enhanced KGs via Rewriting

Ognjen Savković1, Evgeny Kharlamov2, and Steffen Lamparter3

1 Free University of Bozen-Bolzano, Bolzano, Italy
2 University of Oslo, Oslo, Norway
3 Siemens CT, Siemens AG, Munich, Germany

Abstract. Constraints have traditionally been used to ensure data quality. Recently, several constraint languages, among which the W3C recommendation SHACL, have been proposed for Knowledge Graphs (KGs), together with validation mechanisms. An important feature of KGs is that they are often enhanced with ontologies that define relevant background knowledge in a formal language such as OWL 2 QL. At the same time, existing systems for constraint validation either ignore these ontologies, or compile ontologies and constraints into rules that should be executed by some rule engine. In the latter case, one has to rely on different systems when validating constrains over KGs and over ontology-enhanced KGs. In this work, we address this problem by defining rewriting techniques that allow to compile an OWL 2 QL ontology and a set of SHACL constraints into another set of SHACL constraints. In particular, we show that in the general case of OWL 2 QL and SHACL the rewriting may not be possible, and establish restrictions to these languages for which rewriting always exists and also establish complexity bounds as well as identify tractable cases. Our rewriting techniques allow to validate constraints over KGs with and without ontologies using the same SHACL validation engines.

1 Introduction

Constraints has traditionally been used to ensure quality of data in relational [5] and semi-structured databases [4]. Recently constraints have attracted a considerable attention in the context of graph data [18, 17], and in particular for Knowledge graphs (KGs) (e.g., [27, 30, 25]), that is, large collections of interconnected entities that are annotated with data values and types [6]. KGs have become powerful assets for enhancing search and data integration and they are now widely used in both academia and industry [29, 1, 2, 21, 22]. Prominent examples of constraint languages for KGs include SHACL [23]4, ShEx5; examples of constraint validation systems include Stardog6 and TopBraid7.

KGs are often enhanced with OWL 2 ontologies [3] that capture the relevant domain background knowledge with axioms over the terms from the KGs vocabulary e.g., by assigning attributes to classes, defining relationships between classes, composing classes,

4https://www.w3.org/TR/shacl/
5https://www.w3.org/2001/sw/wiki/ShEx
6https://www.stardog.com/
7https://www.topquadrant.com/technology/shacl/
class hierarchies, etc. We will refer to ontology enhanced KGs as Knowledge Bases, or KBs. Ontologies significantly impact constraint validation over KGs. Indeed, constraints over KGs have Closed-World semantics, or Assumption (CWA) in the sense that their validation over a KG boils down to checking whether sub structures of the KG comply with the patterns encoded in the constraints [9, 16, 12]. On the other hand, KBs have open-world semantics (OWA) in the sense that ontologies allow to derive information from a KG that is not explicitly there.

As a result, constraint validation over KGs in the presence of ontologies requires to bridge the CWA of constraints and OWA of ontologies [25, 30, 20]. A promising semantics that offers such bridge was proposed in [25]: given a set of constraints \( C \), an ontology \( \mathcal{O} \), and a KG \( \mathcal{G} \), validating the KB \( \langle \mathcal{O}, \mathcal{G} \rangle \) against \( C \) requires validating all first-order logic models of \( \mathcal{O} \) and \( \mathcal{G} \) that are set-inclusion minimal against \( C \). In practice this can be implemented via a rewriting mechanism: in order to validate \( \langle \mathcal{O}, \mathcal{G} \rangle \) against \( C \), one can compile \( \mathcal{O} \) and \( \mathcal{C} \) into a (possibly disjunctive) logic program and then evaluate the program over \( \mathcal{G} \) [25, 20]. A disadvantage of such approach is that constraint validation in presence of ontologies requires a different evaluation engine than in their absence: it requires an engine for disjunctive logic programs, rather than an engine for validating graph constraints. It is preferable to have a mechanism that allows to evaluate constraints over KBs using the same engine as over KGs.

In this work we address this issue. We first formally formulate the problem of constraints rewriting over ontologies: we require that the result of rewriting is again a set of constraints \( C' \) in the same formalism as the original \( C \). We then study the existence of such a rewriting function for the constraint language SHACL and OWL 2 QL, commonly used profile of OWL 2. Our results show that rewriting may not exist in the general case unless CO-NP = NP, since constraint validation in presence of ontologies is CO-NP-complete, while in absence it is NP-complete.

We next turn our attention to restrictions, on the one hand, of the SHACL language and, on the other hand, of OWL QL. We do this by observing that a source of non-existence of the rewiring function comes from the combination of two factors: (1) the capacity of OWL QL to express the existence of entities that are not explicitly mentioned in a knowledge graph, and (2) the capacity of SHACL to express negation. We then study how elimination of the first factor for OWL 2 QL or the second one for SHACL solves the rewriting problem and propose a corresponding rewriting algorithms. Moreover, we show that if both factors are eliminated for OWL 2 QL, then constraint validation in presence of constraints can be performed in polynomial time, and show both a PTIME rewriting algorithm as well as a PTIME constraint validation algorithm.

The paper is organised as follows: in Section 2 we give preliminaries. In Section 3 we introduce our notion of constrain rewriting over ontologies, and show negative results for SHACL and OWL 2 QL. In Section 4 we present our rewriting algorithms for restricted SHACL constrains and restricted OWL 2 QL ontologies. In Section 5 we put all complexity results together. In Section 6 we discuss related work and extensions our results to other ontology languages. In Section 7 we conclude.

---

8 We note that elimination of the first factor brings OWL 2 QL close to the W3C recommended ontology language RDFS [6] and thus the resulting language is still useful in practice.
2 Preliminaries and Running Example

In this section we recall required definitions. We assume a signature $\Sigma$ that consists of three infinite countable sets of constants, that correspond to entities, classes or unary predicates, that correspond to types, and properties or binary predicates that correspond to object properties or a special predicate “a” that essentially labels entities with classes. Note that we consider neither datatypes nor data properties in this work and leave them for the future study. We also consider an infinite countable domain $\Delta$ of entities.

2.1 Knowledge Graph

A Knowledge Graph (KG) $G$ in our work is a possibly infinite directed label graph that consists of triples of the form $(s, p, o)$ over $\Sigma$, where $s$ is a constant, $p$ is a property, and $o$ is either a constant or a class and only in the latter case $p$ is the special predicate “a”.

Example 1. Consider the following fragment of the Siemens KG $G_{SIEM}$ from [22], which describes Siemens industrial assets including two turbines with the identifiers :t177 and :t852 and one power plant (PPlant) with the identifier :p063, as well as information about equipment (turbine) categories (hasTuCat, hasCat), their deployment sites (deplAt), and enumeration of turbines at plants (hasTurb):

\[
\begin{align*}
&\{(\text{:p063, a, :PPlant}), (\text{:p063, :hasTurb, :t852}), \\
&(\text{:t852, a, :Turbine}), (\text{:t852, :deplAt, :p063}), (\text{:t852, :hasCat, :SGT-800}), \\
&(\text{:t177, :deplAt, :p063}), (\text{:t177, :hasTuCat, :SGT-800})\}.
\end{align*}
\]

2.2 SHACL Syntax

We next briefly recall relevant notions of SHACL using a compact syntax of [12] which is equivalent to SHACL’s “Core Constraint Components” [12]. SHACL stands for Shapes Constraint Language. Each SHACL constraint in a set of constraints $C$, usually referred to as shape, is defined as a triple: $\langle s, \tau_s, \phi_s \rangle$, where

- $s$ is the name,
- $\tau_s$ is the target definition, a SPARQL query with one output variable whose purpose is to retrieve target entities of $s$ from $G$, i.e., entities (nodes) occurring in $G$ for which the following constraint of the shape should be verified,
- and $\phi_s$ is the constraint, an expression defined according to the following grammar:

\[
\phi ::= \top | s' | c | \phi_1 \land \phi_2 | \neg \phi | \geq_n r.\phi | \text{EQ}(r_1, r_2), \tag{1}
\]

where $\top$ stands for the Boolean truth values, $s'$ is a shape name occurring in $C$, $c$ is a constant, $r$ is a SPARQL property path, and $n \in \mathbb{N}^+$; moreover, $\land$ denotes the conjunction, $\neg$ – negation, “$\geq_n r.\phi$” – “must have at least $n$-successors in $G$ verifying $\phi$”, and “EQ($r_1$, $r_2$)” – “$r_1$ and $r_2$-successors of a node must coincide”.

One may also use $\lor$ and $\leq_n r.\phi$ as syntactic sugar, with their expected meaning.
With a slight abuse of notation we identify the shape with its name. We note that the syntax for constraints allows for shapes to reference each other. We call a set of constraints recursive if it contains a shape that reference itself, either directly or via a reference cycle.

**Example 2.** Consider \( C_{SIEM} = \{ \langle s_i, \tau_i, \phi_i \rangle \mid i = 1, 4 \} \), where:

\[
\begin{align*}
\tau_{s_1} &= \exists y (\text{deplAt}(x, y)), & \phi_{s_1} &= \left( \geq 1 :\text{hasCat}, \top \right), \\
\tau_{s_2} &= \exists y (\text{hasTuCat}(x, y)), & \phi_{s_2} &= \left( \geq 1 \text{ a. :Turbine} \right), \\
\tau_{s_3} &= :\text{PPlant}(?x), & \phi_{s_3} &= \left( \geq 1 :\text{hasTurb}, s_4 \right), \\
\tau_{s_4} &= :\text{Turbine}(?x), & \phi_{s_4} &= \left( \geq 1 :\text{deplAt}, s_3 \right).
\end{align*}
\]

Here \( s_1 \) essentially says that any deployed artefact should have a category, and \( s_2 \) says that only turbines can have a turbine category. The last two shapes \( s_3 \) and \( s_4 \) are mutually recursive, and they respectively say that each power plant should have at least one turbine and each turbine should be deployed in at least one location. \( \square \)

### 2.3 SHACL Semantics

Given a shape \( s \), a KG \( G \), and an entity \( e \) occurring \( G \), we say that \( e \) verifies \( s \) in \( G \) if the constraint \( \phi_s \) applied to \( e \) is valid in \( G \). Finally, \( G \) is valid against \( C \) if for each \( s \in C \), each target entity retrieved by \( \tau_s \) from \( G \) verifies \( s \) in \( G \). Since a constraint \( \phi_s \) may refer to a shape \( s' \), the definition of validity for KGS is non-trivial. Indeed, the SHACL specification leaves the difficult case of recursion up to the concrete implementation\(^\text{10}\) and a formal semantics via so-called shape assignments has only recently been proposed [12].

Intuitively, \( G \) is valid against \( C \) if one can label its entities with shape names, while respecting targets and constraints. A shape assignment \( \sigma \) is a function mapping each entity of \( G \) to a set of shape names in \( C \). We call an assignment target-compliant if it assigns (at least) each shape to each of its targets, constraint-compliant if it complies with the constraints, and valid if it complies with both targets and constraints. Then, \( G \) is valid against \( C \) if there exists a valid assignment for \( G \) and \( C \).

**Example 3.** Observe that \( G_{SIEM} \) is not valid against \( C_{SIEM} \). Shape \( s_1 \) has targets :t852 and :t177, since both are deployed. :t852 satisfies the constraint for \( s_1 \), since it has a category, but :t177 violates it. Shape \( s_2 \) has target :t177 only, which violates it, since it is not declared to be a turbine. Shape \( s_3 \) has no target in \( G_{SIEM} \). The case of shape \( s_4 \) is more involved. It has only :t852 as target, and one may assign \( s_4 \) to :t852 and \( s_3 \) to :p063, in order to satisfy the recursive constraint. But since :t177 violates \( s_1 \) and \( s_2 \), there is no "global" valid shape assignment for \( G \) and \( S \), i.e. which would satisfy all targets and constraints simultaneously. \( \square \)

### 2.4 OWL 2 QL

We now recall the syntax and semantics of OWL 2 QL relying on the the Description Logics DL-Lite\(R \) [11] that is behind this profile. (Complex) classes and properties in OWL 2 QL are recursively defined as follows:

\[
B ::= A \mid \exists R, C ::= B \mid \neg B, \ R ::= P \mid P^\rightarrow, \text{ and } E ::= R \mid \neg R,
\]

\(^{10}\) [https://www.w3.org/TR/shacl/](https://www.w3.org/TR/shacl/)
where $A$ is a class from $\Sigma$, $P$ a property from $\Sigma$, and $P^-$ the inverse of $P$. A DL-Lite$_R$ ontology is a finite set of axioms of the form $B \sqsubseteq C$ or $R \sqsubseteq E$. A Knowledge Base (KB) is a pair \(\langle O, G \rangle\) of an ontology and a KG. The formal semantics of DL-Lite$_R$ is given in terms of standard first-order logic interpretations $I = (\Delta, \cdot | I)$ over $\Delta$ in the standard way.

**Example 4.** Consider the following OWL 2 QL ontology $O_{SIEM}$:

\[
\{ :hasTuCat \sqsubseteq :hasCat, \exists :hasTuCat.\top \sqsubseteq :Turbine \},
\]

that says if $x$ has $y$ as a turbine category, then $x$ has $y$ as a category, and also $x$ can be inferred to be a turbine.

A useful property of DL-Lite$_R$ exploited in Section 4, is the existence, for any satisfiable KB $\langle O, G \rangle$, of a so-called canonical model, which can be homomorphically mapped to any model of $\langle O, G \rangle$.

### 2.5 Constraint Validation over Ontology-Enhanced KGs

Consider the semantics of [25], that naturally extends constraint validation from KGs to ontology-enhanced KGs and has been adopted in, e.g., [20]. Given a KG $G$, ontology $O$, and a set of constraints $C$, the idea of this semantics is to validate $C$ over all set inclusion minimal models of $G$ and $O$. Formally, $G$ enhanced with $O$ is valid against $C$ if for each minimal model $M$ of $G$ with $O$, the KG $skol(M)$ is valid against $C$, where $skol(M)$ is the Skolemization of models.

**Example 5.** Observe that $\langle O, G \rangle$ is valid against $C_{SIEM}$. Indeed, shape $s_1$ is still satisfied by :t852, since no new information can be entailed about :t852 from $\langle O_{SIEM}, G_{SIEM} \rangle$. But it is not violated anymore by :t177, since $\langle O_{SIEM}, G_{SIEM} \rangle$ entails that :t177 has a category. Similarly, shape $s_2$ is not violated anymore by :t177, which can now be inferred to be a turbine. As for shape $s_4$, it now has an additional target (:t177), and it is verified by both its targets, thanks to the following assignment:

\[
\{ s_1 \mapsto \{ :t852, :t177 \}, s_2 \mapsto \{ :t177 \}, s_3 \mapsto \{ :p063 \}, s_4 \mapsto \{ :t852, :t177 \} \}.
\]

### 3 The Problem of Constraint Rewriting

We now formalise and discuss the problem of constraint rewriting over ontologies.

#### 3.1 SHACL-Rewriting

Our notion adapts the notion of rewriting (or reformulation) of queries over ontologies from [11, 19].

**Definition 1.** Let $C$ be a set of constraints and $O$ an ontology. A set of constraints $C'$ is a constraint-rewriting of $C$ over $O$ if for any KG $G$ it holds that:

$\langle O, G \rangle$ is valid against $C$ iff $G$ is valid against $C'$.

We now illustrate this notion on the following example.
Example 6. Consider a set of SHACL constraints and an OWL 2 QL ontology:
\[ C = \{(s, \tau_s, \phi_s)\}, \text{ where } \tau_s = :MechDevice(x) \text{ and } \phi_s = (\geq_1 :hasModel. T), \]
\[ O = \{ :Turbin \sqsubseteq :MechDevice, 3 :hasTuCat \sqsubseteq \exists :hasCat \}. \]

One can show that a rewiring of \( C \) over \( O \) is \( S' = \{(s, \tau'_s, \phi'_s)\} \), where
\[ \tau'_s = :MechDevice(x) \lor :Turbin(x) \text{ and } \phi'_s = (\geq_1 :hasCat. T) \lor (\geq_1 :hasTuCat. T). \]

Observe that in the example both the target definition \( \tau_s \) and the constraint definition \( \phi_s \) were rewritten over \( O \) in order to guarantee that the ontology \( O \) can be safely ignored. In particular, the rewriting of \( \tau_s \) guarantees that in any graph \( G \), each instance of \( :Turbin \) should also be verified against \( s \), whereas the rewriting of \( \phi_s \) guarantees that any entity in \( G \) with a \( :hasTuCat - \text{successor} \) validates \( s \), even if it has no \( :hasCat - \text{successor} \).

Thus, despite the similarity of query and constraint rewriting over ontologies there are significant differences. The first difference as illustrated above is that a shape contains a target definition and a constraint that in the general case should be rewritten independently. But more importantly, as opposed to queries, SHACL constraints can be recursive which makes the rewriting significantly more involved (see Section 4 for details).

In what follows we study rewritability for SHACL, i.e. SHACL-rewritability, for different fragments of SHACL. Before proceeding we show that in the general case rewriting does not exist.

3.2 Non-Existence of SHACL-Rewritings

We start with the hardness of SHACL validation.

**Theorem 1.** There exists an DL-Lite\( _R \) ontology, a set of SHACL constraints \( C \), and a KG \( G \) such that deciding whether \( (O, G) \) is valid against \( C \) is co-NP-hard in the size of \( G \).

**Proof (Sketch).** The proof is based on encoding the 3-coloring co-problem. For a given undirected graph \( F = (V, E) \) (with vertices \( V \) and edges \( E \)), we construct the following KG \( G_F \):

\[
\{(v_i, a, V) \mid v_i \in V\} \cup \{(v_i, E, v_j) \mid (v_i, v_j) \in E\} \\
\cup \{(v', U, v_i) \mid v_i \in V\} \cup \{(v', a, T)\},
\]

where \( v', U \) and \( T \) are needed for technical reasons as will be explained below.

Then, we define \( O = \{ V \sqsubseteq \exists R.C, C_{red} \sqsubseteq C, C_{blue} \sqsubseteq C, C_{red} \sqsubseteq \neg C_{blue} \} \), where the axiom \( V \sqsubseteq \exists R.C \) enforces that in each minimal model \( M \) of \( (O, G_F) \), each vertex \( v_i \) has an \( R \)-successor \( a_i \), which intuitively stands for the color of vertex \( v_i \) in \( F \).\(^{12} \)

---

\(^{11}\) Recall that for query rewriting the input is a query \( q \) and ontology \( O \) and the output is another query \( q' \) such that for any database \( D \) so-called certain answers of \( q \) over \( (O, D) \) coincide with the answers of \( q' \) over \( D \) alone [11].

\(^{12}\) The axiom of kind \( V \sqsubseteq \exists R.C \) in syntactically not in DL-Lite\( _R \) but can be expressed by using fresh role \( R_1 \) and three axioms: \( V \sqsubseteq \exists R_1, R_1 \sqsubseteq R \) and \( \exists R_1^- \sqsubseteq C \).
two other axioms intuitively enforce that either \((a_i, a, C_{\text{red}}) \in M\) or \((a_i, a, C_{\text{blue}}) \in M\), or none of the two. Intuitively, \(v_i\) is either red, or blue or none of the two (i.e. green).

Now we introduce a singleton set of constraints \(C = \{\langle s, \tau_s, \phi_s \rangle\}\) that requires that at least one pair of adjacent vertices has the same color:

\[
\tau_s = T(x), \quad \text{and} \quad \phi_s = (\geq 1 U. (\phi_1 \lor \phi_2 \lor \phi_3)),
\]

where
\[
\phi_1 = (\geq 1 R. \geq 1 a. C_{\text{red}}) \land (\geq 1 E. \geq 1 R. \geq 1 a. C_{\text{red}})
\]
\[
\phi_2 = (\geq 1 R. \geq 1 a. C_{\text{blue}}) \land (\geq 1 E. \geq 1 R. \geq 1 a. C_{\text{blue}})
\]
\[
\phi_3 = (\geq 1 R. \geq 1 a. (-C_{\text{red}} \land -C_{\text{blue}})) \land (\geq 1 E. \geq 1 R. \geq 1 a. (-C_{\text{red}} \land -C_{\text{blue}})).
\]

Intuitively, formula \(\phi_1\) evaluates to true at node \(v_i\) node if \(v_i\) is coloured red and has a red neighbour. Formulas \(\phi_2\) and \(\phi_3\) evaluate similarly, but for blue and green. Finally, shape \(s\) has node \(v'\) as a unique target, and \(v\) has every other node in \(G_F\) as a \(U\)-successor, ensuring that \(G_F\) is valid against \(C\) iff there is no 3-colouring for \(F\).

In [13] it has been shown that validation of SHACL constraints over KG without ontologies is NP-complete in the size of the graph. Thus, we can immediately conclude the following negative result that holds under the assumption that \(\text{CO-NP} \not\subseteq \text{NP}\).

**Corollary 1.** There exists an DL-Lite\(_R\) ontology and a set of SHACL constraints for which no SHACL-rewriting over this ontology exists.

In order to overcome the non-existence problem for OWL 2 QL we found possible restrictions on both the ontology language and SHACL (we leave the study of restrictions for OWL 2 EL for future work). In particular, a combination of DL-Lite\(_R\) axioms of the form \(A \sqsubseteq \exists R\) on the one hand, and SHACL constraints with negation on the other hand is sufficient to show that a SHACL-rewriting may not exist. In the next section we investigate rewritability for fragments where such combinations are ruled out.

### 4 Shape Rewriting

In this section we introduce our rewriting algorithms. As discussed above, a rewriting may not exist for an arbitrary set of SHACL shapes and a DL-Lite\(_R\) ontology. Thus to gain rewritability, one needs to restrict the expressivity of either SHACL or the ontology language.

In Section 4.1, we define DL-Lite\(_R^-\), the fragment of DL-Lite\(_R\) where extensional quantification (i.e. concepts of the form \(\exists R\)) is not allowed on the right-hand side of a general concept inclusion (GCI). DL-Lite\(_R^-\) is interesting from a practical point of view, since it corresponds to RDFS. We also define positive SHACL as the fragment of SHACL without negation (but where disjunction is allowed). For DL-Lite\(_R^-\) and whole SHACL we developed a rewriting algorithm Algorithm 1.

In Section 4.2 we restrict SHACL to positive SHACL shapes while we do not restrict DL-Lite\(_R\). For this setting we develop Algorithm 2 that allows to compute constraint rewritings. Note that this algorithm is more involved that Algorithm 1 since DL-Lite\(_R\) KBs may have canonical models of arbitrary size (even infinite).

We observe that the PERFREF algorithm defined in [11] to rewrite (unions of) conjunctive queries in the presence of a DL-Lite\(_R\) ontology cannot be applied in general.
to SHACL shapes, since they may be recursive or contain negations. We nonetheless use PERFRef below when applicable (in particular, for rewriting target definitions). If $O$ is a $DL-Lite_R$ ontology and $\psi$ a conjunctive query (resp. if $C$ is a $DL-Lite_R$ concept) then $\text{PERFRef}(\psi, O)$ (resp. $\text{PERFRef}(C, O)$) designates the rewriting of $\psi$ (resp. the conjunctive query corresponding to $C$) w.r.t. to $O$.

In order to make notation more concise, we use $\mathcal{G}, C \models \phi(v)$ below to indicate that node $v$ satisfies constraint $\phi$ in graph $\mathcal{G}$ given a set $C$ of shapes. Similarly, we use $(\mathcal{O}, \mathcal{G}), C \models \phi(v)$ to indicate that node $v$ satisfies constraint $\phi$ in graph that corresponds to the canonical model $(\mathcal{O}, \mathcal{G})$ given a set $C$ of shapes.

We also assume w.l.o.g. that shape constraints in $C$ are normalized, i.e. contain at most one operator, which can always be obtained by introducing nested shape names.

### 4.1 Rewriting for $DL-Lite_R^-$

In this section, we introduce a rewriting algorithm for $DL-Lite_R^-$ ontologies and full SHACL. We observe that a $DL-Lite_R^-$ KB $(\mathcal{O}, \mathcal{G})$, because it has no existential quantifier on the right-hand side of a GCI, must have a canonical model $\text{can}(\mathcal{O}, \mathcal{G})$ of finite size. This makes rewriting significantly simpler than for arbitrary $DL-Lite_R$ ontologies (investigated in Section 4.2).

Given an ontology $\mathcal{O}$ and set $C$ of shapes, the rewriting $C'$ of $(\mathcal{O}, C)$ is the union of the two sets of shapes: a set $C_O$ built out of $\mathcal{O}$, which is used to mimic ontological reasoning, and a set $C''$ built out of $C$ and $\mathcal{O}$, obtained by (i) rewriting target definitions w.r.t. $\mathcal{O}$, and (ii) rewriting constraints w.r.t. $C$ and $\mathcal{O}$.

**Constructing $C_O$.** For every concept of the form $A$ (resp. $\exists R$) in $\mathcal{O}$, we introduce a shape $s_A$ (resp. $s_{\exists R}$), with no target (i.e. $\tau_{s_A} = \tau_{s_{\exists R}} = \bot(x)$) and with constraint:

$$
\phi_{s_A} = (\exists a.A) \lor \bigvee_{C \subseteq A \in \mathcal{O}} s_C, \quad \phi_{s_{\exists R}} = (\exists_1 R. T) \lor \bigvee_{C \subseteq \exists R \in \mathcal{O}} s_C \lor \bigvee_{R' \subseteq R \in \mathcal{O}} s_{\exists R'},
$$

where $R, R'$ may be inverse roles.

Further, we introduce shapes that encode negative assertions. For each GCI of the form $(C \subseteq \neg D)$ in $\mathcal{O}$, we introduce one shape $s_{C \subseteq \neg D}$, whose targets are all instances of $C$ and $D$ in $\mathcal{G}$, and whose constraint is always violated. To this end, we exploits results based on $\text{PERFRef}$ (see [11]). Namely $\tau_{s_{C \subseteq \neg D}} = \text{PERFRef}(C(x) \land \neg D(x), \mathcal{O})$, and $\phi_{s_{C \subseteq \neg D}} = \bot$. Similarly, for negative role inclusions, we use $s_{R \subseteq \neg R_2}$, with $
\tau_{s_{R \subseteq \neg R_2}} = \text{PERFRef}(\exists y R_1(x, y) \land \neg R_2(x, y), \mathcal{O})$, and $\phi_{s_{R \subseteq \neg R_2}} = \bot$.

We denote the set of shapes produced above with $C_O$, and the corresponding translation function as $\text{SHAPET}$, i.e., $\text{SHAPET}(\mathcal{O}) = C_O$. Then the following holds:

**Lemma 1 (Finite Canonical model).** Let $\mathcal{O}$ be a $DL-Lite_R$ ontology, $C$ a $DL-Lite_R$ concept, $\mathcal{G}$ a graph, $v$ a node in $\mathcal{G}$, $\text{can}(\mathcal{O}, \mathcal{G})$ the canonical model of $\mathcal{O}$ and $\mathcal{G}$. Then $\mathcal{G}, C_0 \cup C'' \models \phi_{s_C}(v)$ iff $(v, a, C) \in \text{can}(\mathcal{O}, \mathcal{G})$.

Next, we rewrite the shapes in $C$. If $s$ is an initial shape in $C$, its rewriting will be designated with $s'$. As already illustrated, both target definitions and constraints need to be rewritten. We start with target definitions.

**Rewriting of Constraints.** In the absence of ontology, the targets of $s$ are retrieved by evaluating the target definition $\tau_s$ over graph $\mathcal{G}$, written $[\tau_s]^{\mathcal{G}}$. In SHACL, a target
Algorithm 1 CONSTRAINT REWRITING FOR DL-Lite⁻

Input: DL-Lite⁻ ontology $\mathcal{O}$, set $\mathcal{C}$ of shapes
1: $C_0 \leftarrow \text{SHAPE}(\mathcal{O})$
2: $C'' \leftarrow \{ \langle s', \text{PERFREF}(\tau_s, \mathcal{O}), \text{REWRITE}(\phi_s, \mathcal{O}) \rangle \mid s \in C \}$
3: return $C_0 \cup C''$

The definition is a monadic query with a single atom that corresponds to a basic concept in an ontology.

In the presence of an ontology, we follow the semantics described in Section 2.5, and retrieve targets over all minimal models, or equivalently over the canonical model, written $[\tau]\langle O, \phi \rangle$. To achieve this, since $\tau_s$ is a unary conjunctive query, one can apply PERFREF.

Lemma 2. For any shape $s$, ontology $\mathcal{O}$ and graph $\mathcal{G}$:
$[\tau_s]\langle O, \phi \rangle = [\text{PERFREF}(\tau_s, \mathcal{O})]\langle \mathcal{G} \rangle$

Proof (Sketch). Follows from the properties of the canonical model and query answering over DL-Lite⁻ ontologies (see [11]).

In other words, the targets of $s$ according to the KB $\langle O, \mathcal{G} \rangle$ can be retrieved by evaluating the query $\text{PERFREF}(\tau_s, \mathcal{O})$ over $\mathcal{G}$ alone.

Rewriting of Constraints. Finally, we rewrite the constraints in $\mathcal{C}$. We replace each shape $s$ by shape $s'$ such that:

- $\phi_{s'} = s'_1 \land s'_2$ if $\phi_s = s_1 \land s_2$,
- $\phi_{s'} = s'_1 \lor s'_2$ if $\phi_s = s_1 \lor s_2$,
- $\phi_{s'} = \neg s'_1$ if $\phi_s = \neg s_1$,
- $\phi_{s'} = I$ if $\phi_s = I$,
- $\phi_{s'} = (\geq_k R.s'_1) \lor \bigvee_{R' \subseteq R \in \mathcal{O}} (\geq_k R'.s'_1)$ if $\phi_s = (\geq_k R.s_1)$,
- $\phi_{s'} = \text{EQ}(R'_1, R'_2)$ if $\phi_s = \text{EQ}(R_1, R_2)$.

Theorem 2. Given a DL-Lite⁻ ontology $\mathcal{O}$, graph $\mathcal{G}$, node $v$ in $\mathcal{G}$, set $\mathcal{C}$ of shapes and shape $s$ in $\mathcal{C}$:
$\langle O, \mathcal{G}, v \rangle \models \phi_s(v)$ iff $\mathcal{C}_0, \mathcal{C}'' \cup \mathcal{C}'' \models \phi_{s'}(v)$

We denote with REWRITE this constraint rewriting function, i.e. $\text{REWRITE}(\phi_s, \mathcal{O}) = \phi_{s'}$ for each $s$ in $\mathcal{C}$.

Algorithm 1 summarize the whole (ontology and shape) rewriting procedure. As an illustration, the SHACL-rewriting described in Example 6 is the one produced by this procedure.

4.2 Rewriting for DL-Lite⁻

We now consider the case of an arbitrary DL-Lite⁻ $\mathcal{O}$, together with a set $\mathcal{C}$ of positive shapes. For any graph $\mathcal{G}$, $\text{can}(\mathcal{O}, \mathcal{G})$ may now be arbitrary large (even infinite) and it may introduce fresh nodes that may be needed to check constraint. This makes rewriting significantly more involved.
Example 7. Consider the ontology \( O = \{ A \sqsubseteq \exists U, \exists U^- \sqsubseteq \exists P \} \) and graph \( G = \{(v, a, A)\} \). Then \( \text{can}(O, G) = \{(v, a, A), (v, U, a_1), (a_1, P, a_2)\} \) where \( a_1 \) and \( a_2 \) are fresh nodes. Now consider shapes \( \langle s_1, A(x), (\geq_1 U, s_2) \rangle \) and \( \langle s_2, \bot(x), (\geq_1 P, \top) \rangle \). It is not hard to see that \( C \) is valid over \( (O, G) \).

Since SHACL constraints cannot express fresh values, we need to introduce additional shapes that mimic the construction of the canonical model.

We start by introducing additional technical notions. Let \( \text{cl}(O, G) \) be the maximal subset of \( \text{can}(O, G) \) which contains no fresh node. We observe that facts in \( \text{cl}(O, G) \) can be validated using \( C \cup \text{SHAPET}(O) \). Now we need to introduce additional shapes to validates facts in \( G' = \text{can}(O, G) \setminus \text{cl}(O, G) \). Graph \( G' \) is in fact a forest (follows from the construction of \( \text{can}(O, G) \)) where each tree has a root in some assertion in \( \text{cl}(O, G) \).

We call this root the witness of the tree. In example above, \( (v, a, A) \) is the only witness.

Then for each concept \( C \) appearing in a GCI in \( O \), we introduce a shape \( s^\text{virtual}_C \), such that, for a node \( v \) in \( G \), verifies \( s^\text{virtual}_C(v) \) if is there a node \( v' \) in \( G' \) with \( v \) as witness such that \( G' \models C(v') \). For instance, in Example 7, we introduce shape \( s^\text{virtual}_U \) which is verified by witness \( v \). Note that from this definition, \( v' \) is not necessarily an immediate successor of \( v \) in \( G' \).

More formally, for concept \( C \), the virtual shape \( \langle s^\text{virtual}_C, \bot(x), s_C \rangle \) is created. Then a function similar to \( \text{REWITERSIM} \) is applied to each \( \phi^\text{virtual} \), in order to ensure the above property. In our running example, this yields \( \phi^\text{virtual}_{s_A} = s_A \), i.e. \( \phi^\text{virtual} \) remains unchanged, but \( \phi^\text{virtual}_{s^\text{virtual}_A} = s^\text{virtual}_A \). Here, sub-formula \( s^\text{virtual} \) is added because of the GCI \( A \sqsubseteq U \), and \( s^\text{virtual}_U \) is added because if \( \exists U \) holds at some node \( a_1 \) in the tree of \( G' \) rooted in \( v \), then \( \exists U^− \) must hold at some \( U \)-successor \( a_2 \) of \( a_1 \). Let \( \text{SHAPEVIRTUAL} \) be the function which produces (and rewrites) these “virtual” shapes.

A second kind of shape is needed in order to check if two roles are concatenated in the same tree in \( G' \). For each pair of roles \( R_1 \) and \( R_2 \) in \( O \), we introduce shape \( s^\text{succ}_{R_1, R_2} \) such that a node \( v \in G \) verifies \( \phi^\text{succ}_{s^\text{virtual}} \) iff \((a_1, R_1, a_2)\) and \((a_2, R_2, a_3)\) are on the subtree with witness \( v \), for some \( a_1, a_2, a_3 \) in \( G^\prime \). In our running example, \( v \) verifies \( \phi^\text{succ}_{s^\text{virtual}} \), but not \( \phi^\text{succ}_{s^\text{virtual}} \). Formally, for every two roles \( R_1 \) and \( R_2 \) in \( O \), \( \tau_{s^\text{virtual}_{R_1, R_2}} = \bot(x) \), and if \( O \models \exists R_1^- \sqsubseteq R_2 \), then \( \phi^\text{succ}_{s^\text{virtual}_{R_1, R_2}} = s_{\exists R_1} \), otherwise \( \phi^\text{succ}_{s^\text{virtual}_{R_1, R_2}} = \bot \). The special case \( R_2 = R_1^- \) is also covered by the definition \( \phi^\text{succ}_{s^\text{virtual}_{R_1, R_1^-}} = s_{\exists R_1} \). Let \( \text{SUCCESSORT} \) denote the function creating these fresh shapes.

Finally, we need to rewrite the shapes in \( C \). To this end, we extend the procedure \( \text{REWITERSIM} \) in the following way. For each shape \( s \) in \( C \), we set \( s' = \text{REWITEREOMPL}(s) \vee s^\text{virtual} \) where \( \text{REWITEREOMPL} \) is identical to \( \text{REWITERSIM} \) for operators \( \land, \lor \) and constant \( I \) but it changes for \( \phi_s = (\geq_k R, s_1) \) as follows:

\[
\phi'_s = (\geq_k R, s_1') \lor s^\text{virtual} \quad \text{where} \quad \phi^\text{virtual} = s^\text{virtual}_{R} \land s^\text{virtual}_1 \land s_{\exists R, s_1}
\]

In other words, witness \( v \) verifies \( s^\text{virtual} \) if it verifies both \( s^\text{virtual}_{R} \) and \( s^\text{virtual}_1 \) (that is, both are verified by some anonymous node with \( v \) as witness), and the range of \( R \) can be validated against \( s_1 \), expressed with the new shape \( s_{\exists R, s_1} \). Then \( \phi_{s_{\exists R, s_1}} = s_{\exists R, s_2} \land s_{\exists R, s_3} \) if \( s_{\exists R, s_1} = s_2 \land s_3 \) (and similarly for \( \lor \)). If \( s_{\exists R, s_1} = (\geq_k P, s_2) \) then \( \phi_{s_{\exists R, s_1}} = s^\text{virtual}_{P} \), that is \( P \) has to be successor of \( R \) in \( G' \). Let \( \text{REWITEREOMPLT} \) denote the corresponding rewriting of \( C \).
Algorithm 2 CONSTRAINT REWRITING 2

Input: ontology $\mathcal{O}$ possible with existential rules, set of positive shapes $C$
1: $C_0 \leftarrow \text{SHAPE}(\mathcal{O})$
2: $C_\phi \leftarrow \text{SHAPEVIRTUAL}(\mathcal{O})$
3: $C_{\phi, \mathcal{O}} \leftarrow \text{SUCCESSOR}(\mathcal{O}, S)$
4: $C'' \leftarrow \{\text{PERFREP}(\tau, \mathcal{O}), \text{REWRITECOMPL}(\phi_s, \mathcal{O})) \mid s \in C\}$
5: return $C_0 \cup C_\phi \cup C_{\phi, \mathcal{O}} \cup C''$

We summarize the rewriting procedure in Algorithm 2.

Lemma 3. Let $\mathcal{O}$ be a DL-Lite ontology, $C$ a concept in $\mathcal{O}$, $R$ and $P$ properties in $\mathcal{O}$, $\mathcal{G}$ a graph, $v$ a node in $\mathcal{G}$, $\text{can}(\mathcal{O}, \mathcal{G})$ the canonical model of $\mathcal{O}$ and $\mathcal{G}$, $C$ a set of positive shapes and $C'$ the shapes returned by Algorithm 2. Then the following holds:

- $\mathcal{G}, C' \models \phi_{\text{virtual}}(v)$ iff there is a node $a_1$ in $\text{can}(\mathcal{O}, \mathcal{G})$ with witness $v$ s.t. $\text{can}(\mathcal{O}, \mathcal{G}) \models C(a_1)$
- $\mathcal{G}, C' \models \phi_{\text{succ}}(v)$ iff there are nodes $a_1, a_2, a_3$ in $\text{can}(\mathcal{O}, \mathcal{G})$ with witness $v$ s.t. $\text{can}(\mathcal{O}, \mathcal{G}) \models R(a_1, a_2)$ and $\text{can}(\mathcal{O}, \mathcal{G}) \models R(a_2, a_3)$

Example 8. We illustrate the rewriting of the running example. Shapes that are not relevant for reasoning are omitted. The presented shapes are ordered in the way one would reason with them, starting bottom-up (which is possible if $C$ is not recursive). To illustrate the reasoning, we underline in each formula the disjuncts for which one can construct a satisfying shape assignment.

\[
\begin{align*}
\phi_{s_A} &= s_A, \\
\phi_{s_{\exists U, T}} &= s_{\exists U} \lor s_{\exists U, T}, \\
\phi_{s_{\exists U, P}} &= \phi_{s_{\exists U, P}} = s_{\exists U, P}, \\
\phi_{s_{\exists U, T}} &= s_{\exists U, T}, \\
\phi_{s_{\exists U, P}} &= s_{\exists U, P}, \\
\phi_{s_{\exists U, T}} &= s_{\exists U, T}. 
\end{align*}
\]

The only target of $s$ is $v$, and $v$ verifies $\phi_s$ w.r.t the rewritten set of shapes.

Theorem 3. Let $\mathcal{O}$ be a DL-Lite ontology, $C$ a set of positive shapes, $s$ a shape in $C$, $C'$ the shapes returned by Algorithm 2, and $s'$ the rewriting of $s$ in $C'$.

For any graph $\mathcal{G}$ and node $v$ in $\mathcal{G}$

\[\langle \mathcal{O}, \mathcal{G} \rangle, C \models \phi_s(v) \text{ iff } \mathcal{G}, C' \models \phi'_s(v)\]

We also note that the size of the returned rewriting is polynomial in size of $\mathcal{O}$ and $C$.

5 Complexity of Validation

In this section we discuss the complexity of validating a KB against SHACL constraints as summarised in Table 1. These results follow from Sections 3 and 4 and previous results about SHACL constraint validation.
Table 1. Combined complexity of graph validation against a (compact) SHACL-rewriting, for different fragments of SHACL and DL-Lite

<table>
<thead>
<tr>
<th></th>
<th>Full SHACL</th>
<th>Positive SHACL</th>
</tr>
</thead>
<tbody>
<tr>
<td>DL-Lite</td>
<td>DP-hard</td>
<td>PTIME-complete</td>
</tr>
<tr>
<td>DL-Lite₂</td>
<td>NP-complete</td>
<td>PTIME-complete</td>
</tr>
</tbody>
</table>

We start by observing that despite the fact that for full SHACL and $DL-Lite₂$, rewriting in general does not exist, we can still provide a lower-complexity bound on validation of KBs over constraints. In particular, from Theorem 1, validation is at least $\text{CO-NP}$-hard in data complexity. Since it is also at least NP-hard for full SHACL, one can show that the problem is DP-hard [26].

It was shown in [12] that in the case without ontologies validation for the full SHACL language is NP-complete in both data (i.e. the size of the graph) and combined (i.e. graph and constraints) complexity, e.g., due to the combination of recursion and negation. At the same time tractability can be gained by restricting their usage as in positive SHACL, for which the validation is PTIME-complete in both data and combined complexity. Another way to gain the tractability is as [14] where the notion of strict stratification for SHACL constraints was introduced and it strengthen the classical notion of stratification in Datalog. It was shown that validation for such constraints is also in PTIME. In Section 4.1 our rewriting over ontologies of disjointness constraints for $DL-Lite₂$ and positive SHACL is strictly stratified, and this is the only possible source of negation. Since our rewriting is of polynomial size, we can conclude that validating a $DL-Lite₂$ KB against positive SHACL constraints is in PTIME.

6 Discussion And Related work

In this section we discuss what we thinks are interesting properties of SHACL rewritings as well as related work.

6.1 On Compactness of Rewriting

The rewriting technique presented in this paper also have a desirable property of compactness. Compactness of rewritability has been studied query rewriting over OWL 2 QL ontologies [8], and it has been shown (under some separation assumptions) that a polynomial rewriting cannot exist in general for several non-recursive languages (including non-recursive Datalog). In comparison, our techniques take the advantage of recursive shape references to produce a worst-case polynomial SHACL-rewriting.

6.2 General Rewriting Algorithm

In general, rewriting of OWL 2 QL can be seen a special case of general backward-chaining algorithm [24] algorithm over so-called $\forall\exists$-rules (where the body and the head are conjunctions of atoms, and variables that occur only in the head are existentially quantified). The existence of a rewriting is undecidable in general for arbitrary rules, and even for some decidable fragments, this algorithm may not terminate. Nevertheless, since
it is a general one it is a good starting point to investigate terminations and optimizations when applied to more restrictive ontologies.

6.3 Beyond OWL 2 QL

So far we focused on restricting SHACL constraints to gain rewritability. A natural question to ask what if we go beyond DL-LiteR even if for restricted SHACL constraints or consider languages like OWL 2 EL. First, observe that for OWL 2 EL one can prove the non existence of rewriting of constraints over ontologies using the same argument is in Theorem 1. Still, compact rewriting in case of positive SHACL may be possible and we leave it as an open question. Then, consider $\mathcal{ALC}$ [7], a DL that is contained in OWL 2, has been long studied and that properly contains both QL and EL profiles of OWL 2. One can show that $\mathcal{ALC}$ is too expressive so that can be rewritten into SHACL even if we are given a single shape of the formula of the form “$\top$”. For example, one can reduce the problem of “concept satisfiability” in $\mathcal{ALC}$ into checking validity of SHACL constraints of over the same $\mathcal{ALC}$ KB. Since the former is known to be PSPACE-complete [7], we have a problem that cannot be encoded as SHACL validation problem (which is worst case NP-complete). The Datalog$^\pm$ family [10] is another popular family of ontological languages that extend the idea of first-order query rewritability to $\forall\exists$-rules but under certain syntactic restrictions. For Datalog$^\pm$ despite the fact that instance checking is very efficient in data complexity ($A C^0$) meaning that the query answering can be reduced to evaluation of first-order queries, the price of the combined complexity of the problem is rather high (EXPTIME-hard) and thus one cannot obtain polynomial rewritings in SHACL.

6.4 Translating SHACL to known query language

Alternative idea of rewriting SHACL is to relate SHACL in some know ontological (query) language and then apply rewriting algorithm. Since SHACL may contain recursion such query language cannot be of conjunctive queries. Naturally, one may think of relating SHACL to Datalog programs [15]. However, Datalog programs can at most have one unique minimal model, and SHACL constraints are checking for all possible assignments [12] (including also non-minimal one). If we consider more expressive version of Datalog like Datalog with negation under the stable model semantics (SMS) [15], then relating it to SHACL is more promising, while the actual relation is not obvious as SMS is also based on minimal models. A possible way to relate the two semantics, at least for SHACL with single maximal assignments, is to reason about shape “complements” under SMS. Nevertheless, our preliminary results show that this is not straightforward.

7 Conclusion

In this work, we study the problem of rewriting constraints over ontologies. We focused on a prominent language for graph constraints, namely SHACL, and on ontologies from the widely used OWL 2 QL ontology language. We defined semantics for constraint rewriting, showed the non-existence of such rewritings in the general case, and identified restrictions of OWL 2 QL and SHACL for which they always exist. For these restricted cases, we showed how to rewrite ontologies and SHACL into unique set of SHACL constraints. Moreover, validation over OWL 2 QL is tractable for the positive (but still
recursive) fragment of SHACL. And validation with full SHACL expressivity becomes
NP-complete. For the case where the existence of rewritings cannot be guaranteed, we
established lower complexity bound.

We see this work as an important step towards practical constraint rewriting algorithms
and systems. In particular, instead of combining OWL reasoning and SHACL validation
as an additional layer may create an unnecessary overhead, our rewritings allow the
validation that could be performed against a single set shapes. Next, we plan to analyze
optimization techniques in order to obtain more efficient rewritings. For instance, we plan
to consider datatypes. They can be used to optimize eliminate unnecessary rewritings,
but this need to be done in a controlled way to ensure tractability (e.g., [28]). Finally, in
the future we plan to extend this work to account for OWL 2 EL. Moreover, we plan to
implement our approach and evaluate it.

Bibliography

Semistructured Data and XML. Morgan Kaufmann, 1999.
1995.
description logic handbook: theory, implementation and applications. Cambridge
Ontology-mediated queries: Combined complexity and succinctness of rewritings
of Shapes Schemas for RDF. In ISWC, 2017.
datalog+/. In RR.
reasoning and efficient query answering in description logics: The DL-Lite family.
shacl. ISWC, 2018.
shacl (extended version). Technical Report KRDB18-1, KRDB Research Center,
SHACL. In ISWC, 2018.


