Validation of SHACL Constraints over KGs with OWL 2 QL Ontologies via Rewriting

Ognjen Savković[1(✉)], Evgeny Kharlamov[2,3], and Steffen Lamparter[4]

[1] Free University of Bozen-Bolzano, Bolzano, Italy
ognjen.savkovic@unibz.it
[2] University of Oslo, Oslo, Norway
[3] Bosch Centre for Artificial Intelligence, Robert Bosch GmbH, Renningen, Germany
[4] Siemens CT, Siemens AG, Munich, Germany

Abstract. Constraints have traditionally been used to ensure data quality. Recently, several constraint languages such as SHACL, as well as mechanisms for constraint validation, have been proposed for Knowledge Graphs (KGs). KGs are often enhanced with ontologies that define relevant background knowledge in a formal language such as OWL 2 QL. However, 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. We show that in the general case the rewriting may not exists, but it always exists for the positive fragment of SHACL. Our rewriting techniques allow to validate constraints over KGs with and without ontologies using the same SHACL validation engines.

1 Introduction

Constraints have traditionally been used to ensure quality of data in relational [5] and semi-structured DBs [4]. Recently constraints have attracted considerable attention in the context of graph data [16,17], and in particular for *Knowledge Graphs* (KGs) (e.g, [35,36,42]), i.e., large collections of interconnected entities that are annotated with data values and types [7]. KGs have become powerful assets for enhancing search and data integration and they are now widely used in both academia and industry [1,2,6,19,22–25,28,40,41]. Prominent examples of constraint languages for KGs include SHACL [31], ShEx[1]; and of constraint validation systems Stardog[2] and TopBraid[3].

[1] https://www.w3.org/2001/sw/wiki/ShEx.
[2] https://www.stardog.com/.
[3] https://www.topquadrant.com/technology/shacl/.

P. Hitzler et al. (Eds.): ESWC 2019, LNCS 11503, pp. 314–329, 2019.
https://doi.org/10.1007/978-3-030-21348-0_21

KGs are often enhanced with *ontologies*, expressed in, e.g., the OWL 2 ontology language [3]. Ontologies capture the relevant background knowledge with axioms over the terms from the KG's vocabulary e.g., by assigning attributes to classes, by defining relationships between classes, composed classes, and class hierarchies. We refer to ontology enhanced KGs as*Knowledge Bases* (*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 [8,12,15]. 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 [20,21,35,42]. A promising semantics that offers the bridge was proposed in [35]: given a set of constraints $\mathcal{C}$, ontology $\mathcal{O}$, and KG $\mathcal{G}$, validating the KB $\langle \mathcal{O}, \mathcal{G} \rangle$ against $\mathcal{C}$ requires to validate all first-order logic models of $\mathcal{O}$ and $\mathcal{G}$ that are set-inclusion minimal against $\mathcal{C}$. This can be done via a rewriting mechanism: in order to validate $\langle \mathcal{O}, \mathcal{G} \rangle$ against $\mathcal{C}$, one can compile $\mathcal{O}$ and $\mathcal{C}$ into a (possibly disjunctive) logic program and then evaluate the program over $\mathcal{G}$ [20,35]. A disadvantage of this approach is that constraint validation in the 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. However, from practical point of view it is desirable 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 formalise the problem of constraints rewriting over ontologies: we require that the result of rewriting is again a set of constraints $\mathcal{C}'$ in the same formalism as the original $\mathcal{C}$. We then study the existence of such a rewriting function for the constraint language SHACL and the ontology language OWL 2 QL which is 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 consider the restriction of SHACL to positive constraints, that we call SHACL$^+$, and show that in this case the rewriting always exists and provide an algorithm for such rewriting.

2 Preliminaries and Running Example

In this section we recall required definitions. We assume a signature Σ of three infinite countable sets of *constants*, that correspond to entities, *classes* of unary predicates, that correspond to types, and *properties* or binary predicates, that correspond to object properties or a special predicate "a" that labels entities with classes. Note that do not consider datatypes and data properties, and leave them for the future study. We assume an infinite countable *domain* Δ of entities.

2.1 Knowledge Graph

A Knowledge Graph (KG) $\mathcal{G}$ is a possibly infinite directed labeled graph that consists of triples of the form (s, p, o) over Σ, where s is a constant, p – property, and o – constant or class (in this case p is the special predicate "a").

Example 1. Consider the following fragment of the Siemens KG $\mathcal{G}_{SIEM}$ from [23], 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`):

$$\{(\texttt{:p063, a, :PPlant}), (\texttt{:p063, :hasTurb, :t852}), (\texttt{:t852, a, :Turbine}),$$
$$(\texttt{:t852, :deplAt, :p063}), (\texttt{:t852, :hasCat, :SGT-800}),$$
$$(\texttt{:t177, :deplAt, :p063}), (\texttt{:t177, :hasTuCat, :SGT-800})\}.$$ □

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 $\mathcal{C}$, usually referred to as *shape*, is defined as a triple: $\langle s, \tau_s, \phi_s \rangle$, where

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

$$\phi \ ::= \ \top \mid s' \mid c \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \neg\phi \mid \geq_n R.\phi \mid \leq_n R.\phi \mid \mathrm{EQ}(r_1, r_2), \quad (1)$$

  where $\top$ stands for the Boolean truth values, s' is a shape name occurring in $\mathcal{C}$, c is a constant, R is a property, and $n \in \mathbb{N}$; moreover, $\wedge$ denotes the conjunction, $\neg$ – negation, "$\geq_n R.\phi$" – "must have at least n-successors in $\mathcal{G}$ verifying ϕ", r_1 and r_2 are SPARQL property paths and "$\mathrm{EQ}(r_1, r_2)$" means that "r_1 and r_2 successors of a node must coincide".

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. A set of constraints is *recursive* if it contains a shape that reference itself, either directly or via a reference cycle.

Example 2. Consider $\mathcal{C}_{SIEM} = \{\langle s_i, \tau_{s_i}, \phi_{s_i}\rangle \mid i = 1, 4\}$, where:

$$\tau_{s_1} = \exists y(\texttt{:deplAt}(x, y)), \qquad \phi_{s_1} = (\geq_1 \texttt{:hasCat}.\top),$$
$$\tau_{s_2} = \exists y(\texttt{:hasTuCat}(x, y)), \qquad \phi_{s_2} = (\geq_1 \texttt{a.:Turbine}),$$
$$\tau_{s_3} = \texttt{:PPlant}(?x), \qquad \phi_{s_3} = (\geq_1 \texttt{:hasTurb}.s_4),$$
$$\tau_{s_4} = \texttt{:Turbine}(?x), \qquad \phi_{s_4} = (\geq_1 \texttt{:deplAt}.s_3).$$

Here s_1 essentially says that any deployed artifact 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. □

2.3 SHACL Semantics

Given a shape s, a KG $\mathcal{G}$, and an entity e occurring $\mathcal{G}$, we say that e *verifies* s in $\mathcal{G}$ if the constraint ϕ_s applied to e is valid in $\mathcal{G}$. Finally, $\mathcal{G}$ is *valid* against $\mathcal{C}$ if for each $s \in \mathcal{C}$, each target entity retrieved by τ_s from $\mathcal{G}$ verifies s in $\mathcal{G}$. Since a constraint ϕ_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[4] and a formal semantics via so-called *shape assignments* has only recently been proposed [12]. Intuitively, $\mathcal{G}$ is valid against $\mathcal{C}$ if one can label its entities with shape names, while respecting targets and constraints. A shape assignment σ is a function mapping each entity of $\mathcal{G}$ to a set of shape names in $\mathcal{C}$. 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, $\mathcal{G}$ is valid against $\mathcal{C}$ if there exists a valid assignment for $\mathcal{G}$ and $\mathcal{C}$.

Example 3. Observe that $\mathcal{G}_{SIEM}$ is not valid against $\mathcal{C}_{SIEM}$. 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 the target `:t177` only, which violates it, since it is not declared to be a turbine. Shape s_3 has no target in $\mathcal{G}_{SIEM}$. The case of shape s_4 is more involved. It has only `:t852` as the 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 $\mathcal{G}$ and $\mathcal{S}$, i.e. which would satisfy all targets and constraints simultaneously. □

2.4 OWL 2 QL

We now recall the syntax and semantics of OWL 2 QL relying on the the Description Logics *DL-Lite*$_R$ [9] that is behind this profile. (Complex) classes and properties in OWL 2 QL are recursively defined as follows:

$$B ::= A \mid \exists R,\ C ::= B \mid \neg B,\ R ::= P \mid P^-, \text{ and } E ::= R \mid \neg R,$$

[4] https://www.w3.org/TR/shacl/.

where A is a class from Σ, P a property from Σ, and P^- the inverse of P. Expression A we call also an *atomic* class or concepts and B a *basic* class or concepts. 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 $\langle \mathcal{O}, \mathcal{G} \rangle$ of an ontology and a KG. The formal semantics of *DL-Lite*$_R$ is given in terms of first-order logic interpretations $\mathcal{I} = (\Delta, \cdot^{\mathcal{I}})$ over Δ in the standard way.

Example 4. Consider the following OWL 2 QL ontology $\mathcal{O}_{SIEM}$:

$$\{\texttt{:hasTuCat} \sqsubseteq \texttt{:hasCat}, \quad \exists \texttt{:hasTuCat}.\top \sqsubseteq \texttt{:Turbine}\},$$

that says that 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 Sect. 4, is the existence, for any satisfiable KB $\langle \mathcal{O}, \mathcal{G} \rangle$, of a so-called *canonical model*, which can be homomorphically mapped to any model of $\langle \mathcal{O}, \mathcal{G} \rangle$.

2.5 Constraint Validation over KGs Enhanced with Ontologies

Consider the semantics of [35], that naturally extends constraint validation from KGs to ontology-enhanced KGs and has been adopted in, e.g., [20]. Given a KG $\mathcal{G}$, ontology $\mathcal{O}$, and a set of constraints $\mathcal{C}$, the idea of this semantics is to validate $\mathcal{C}$ over all set inclusion minimal models of $\mathcal{G}$ and $\mathcal{O}$. Formally, $\mathcal{G}$ enhanced with $\mathcal{O}$ is *valid* against $\mathcal{C}$ if for each minimal model $\mathcal{M}$ of $\mathcal{G}$ with $\mathcal{O}$, the KG *skol*($\mathcal{M}$) is valid against $\mathcal{C}$, where *skol*($\mathcal{M}$) is the Skolemization of models.

Example 5. Observe that $\langle \mathcal{O}, \mathcal{G}, \rangle$ is valid against $\mathcal{C}_{SIEM}$. Indeed, shape s_1 is still satisfied by `:t852`, since no new information can be entailed about `:t852` from $\langle \mathcal{O}_{SIEM}, \mathcal{G}_{SIEM} \rangle$. Moreover, s_1 is now not violated by `:t177`: $\langle \mathcal{O}_{SIEM}, \mathcal{G}_{SIEM} \rangle$ entails that `:t177` has a category. Similarly, s_2 is now not violated by `:t177`: one can can infer that it is a turbine. The shape s_4 now has an additional target (`:t177`), and it is verified by both its targets, thanks to the following assignment: $\{s_1 \mapsto \{\texttt{:t852}, \texttt{:t177}\}, s_2 \mapsto \{\texttt{:t177}\}, s_3 \mapsto \{\texttt{:p063}\}, s_4 \mapsto \{\texttt{:t852}, \texttt{:t177}\}\}$. □

3 The Problem of Constraint Rewriting

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

3.1 SHACL-Rewriting

In order to define SHACL rewriting we adapt the notion of rewriting (or reformulation) of queries over ontologies from [9,18].

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

$$\langle \mathcal{O}, \mathcal{G} \rangle \textit{ is valid against } \mathcal{C} \textit{ iff } \mathcal{G} \textit{ is valid against } \mathcal{C}'.$$

We now illustrate this notion on the following example.

Example 6. Consider a set of SHACL constraints and an OWL 2 QL ontology:

$$\mathcal{C} = \{\langle s, \tau_s, \phi_s\rangle\}, \text{ where } \tau_s = \texttt{:MechDevice}(x) \text{ and } \phi_s = (\geq_1 \texttt{:hasCat}.\top),$$
$$O = \{\texttt{:Turbine} \sqsubseteq \texttt{:MechDevice}, \exists\texttt{:hasTuCat} \sqsubseteq \exists\texttt{:hasCat}\}.$$

One can show that a rewiring of $\mathcal{C}$ over $\mathcal{O}$ is $\mathcal{S}' = \{\langle s, \tau'_s, \phi'_s\rangle\}$, where

$$\tau'_s = \texttt{:MechDevice}(x) \vee \texttt{:Turbine}(x) \text{ and}$$
$$\phi'_s = (\geq_1 \texttt{:hasCat}.\top) \vee (\geq_1 \texttt{:hasTuCat}.\top).$$

□

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

Thus, despite the similarity of query and constraint rewriting overt ontologies there are significant differences[5]. 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 Sect. 4 for details).

We now show that rewritings may not exist.

3.2 Non-existence of SHACL-Rewritings

We start with the hardness of SHACL validation that can be shown by reduction from the 3-coloring co-problem.

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

Proof. [Sketch] The proof is based on an encoding of the 3-coloring co-problem into the validity problem. For a given undirected graph $\mathcal{F} = \langle V, E\rangle$, where V is a set of vertices and E of edges, we construct the following KG $G_{\mathcal{F}}$:

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

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

[5] Recall that for query rewriting the input is a query q and ontology $\mathcal{O}$ and the output is another query q' such that for any database D so-called certain answers of q over $\langle \mathcal{O}, \mathcal{D}\rangle$ coincide with the answers of q' over $\mathcal{D}$ alone [9].

Then, we define $\mathcal{O} = \{V \sqsubseteq \exists R.C, C_{red} \sqsubseteq C, C_{blue} \sqsubseteq C, C_{red} \sqsubseteq \neg C_{blue}\}$, where the axiom $V \sqsubseteq \exists R.C$ enforces that in each minimal model $\mathcal{M}$ of $\langle \mathcal{O}, \mathcal{G}_{\mathcal{F}} \rangle$, each vertex v_i has an R-successor a_i, which intuitively stands for the color of vertex v_i in $\mathcal{F}$[6]. The two other axioms intuitively enforce that either $(a_i, \mathsf{a}, C_{red}) \in \mathcal{M}$ or $(a_i, \mathsf{a}, C_{blue}) \in \mathcal{M}$, 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 $\mathcal{C} = \{\langle s, \tau_s, \phi_s \rangle\}$ that requires that at least one pair of adjacent vertices has the same color:

$$\tau_s = T(x), \text{ and } \phi_s = (\geq_1 U.(\phi_1 \vee \phi_2 \vee \phi_3)), \text{ where}$$

$$\phi_1 = (\geq_1 R. \geq_1 \mathsf{a}.C_{red}) \wedge (\geq_1 E. \geq_1 R. \geq_1 \mathsf{a}.C_{red})$$
$$\phi_2 = (\geq_1 R. \geq_1 \mathsf{a}.C_{blue}) \wedge (\geq_1 E. \geq_1 R. \geq_1 \mathsf{a}.C_{blue})$$
$$\phi_3 = (\geq_1 R. \geq_1 \mathsf{a}.(\neg C_{red} \wedge \neg C_{blue})) \wedge (\geq_1 E. \geq_1 R. \geq_1 \mathsf{a}.(\neg C_{red} \wedge \neg C_{blue})).$$

Intuitively, the formula ϕ_1 evaluates to true at the node v_i if v_i is colored as red and has a red neighbour. The formulas ϕ_2 and ϕ_3 evaluate similarly, but for blue and green. Finally, the shape s has the node v' as the unique target, and v has every other node in $G_{\mathcal{F}}$ as a U-successor, ensuring that $G_{\mathcal{F}}$ is valid against $\mathcal{C}$ iff there is no 3-colouring for $\mathcal{F}$. □

In [13] it has been shown that validation of SHACL constraints over KGs 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 co-NP $\not\subseteq$ 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 we found a restriction on SHACL as will be presented in the following section.

4 Rewriting of SHACL$^+$ Constraints over OWL 2 QL

As discussed above, a rewriting may not exist for an arbitrary set of SHACL shapes and a *DL-Lite$_R$* ontology. Thus, in order to gain rewritability one can restrict the expressivity of SHACL. In the following we do so by restricting SHACL to positive SHACL shapes. For this setting we develop Algorithm 1 that allows to compute constraint rewritings. A SHACL shape is in SHACL$^+$ if it does not contain negation and cardinality restriction of kind "$\leq_n R.\phi$".

[6] The axiom of the kind $V \sqsubseteq \exists R.C$ in syntactically not in *DL-Lite$_R$* but it can be expressed using a "fresh" role R_1 and three axioms: $V \sqsubseteq \exists R_1$, $R_1 \sqsubseteq R$ and $\exists R_1^- \sqsubseteq C$.

The rest of this section we organize in four parts. *(i)* First we show why it is sufficient to consider only satisfiable KGs; *(ii)* Then we show that over such satisfiable KGs it sufficient to focus only on their canonical models *(iii)* then we show how to rewrite shape targets given an ontology, and *(iv)* finally we show how to rewrite ontologies.

4.1 Satisfiable Knowledge Graphs

We observe that in general KGs may contain disjointness and thus they may be unsatisfiable (that is, they have no model). First we introduce an axillary property that shows that for constraint validation it is sufficient to consider only satisfiable KGs.

Lemma 1. *Let $\mathcal{O}$ be a DL-Lite$_R$ ontology and $\mathcal{G}$ a graph. If $\langle \mathcal{O}, \mathcal{G} \rangle$ is unsatisfiable then for any shape s and any node in v in $\mathcal{G}$ there is exists no satisfying shape assignment over $\mathcal{G}$ and the set of shapes $\mathcal{C}_\mathcal{O}$ that validates $s(v)$.*

Proof. Assume that $\langle \mathcal{O}, \mathcal{G} \rangle$ is unsatisfiable. Wlog we assume that the cause of being unsatisfiable is the following: $\langle \mathcal{O}, \mathcal{G} \rangle \models C(a) \wedge D(a)$ holds for some node a in $\mathcal{G}$ and some basic concepts C and D, and at same time $\mathcal{O} \models C \sqsubseteq \neg D$ holds (similarly it can be shown for role disjointness).

From $\langle \mathcal{O}, \mathcal{G} \rangle \models C(a) \wedge D(a)$ we conclude (using the properties on PerfRef in [9]) that $\mathcal{G} \models \textsc{PerfRef}(C(a) \wedge D(a), \mathcal{O})$. Then, since we have $\tau_{s_{C \sqsubseteq \neg D}} = \textsc{PerfRef}(C(x) \wedge D(x), \mathcal{O})$ it follows that a is the target of the shape $s_{C \sqsubseteq \neg D}$. On the other hand $\phi_{s_{C \sqsubseteq \neg D}} = \bot$, thus for every shape assignment σ it must be $\sigma(s_{C \sqsubseteq \neg D}) = \bot$, i.e., there exists no satisfying assignment for $s_{C \sqsubseteq \neg D}$. Hence, there exist no satisfying assignment over $\mathcal{G}$ and $\mathcal{C}_\mathcal{O}$ that also validates $s(v)$. □

4.2 Validity over Canonical Models

In this part we show that in order to check the validity over all minimal models it is sufficient to check the validity over the canonical model for the given KG. In [9], the authors defined a canonical model of a KB as a model that can be homomorphically mapped to any other model of that KB. Now we extend this notion to shapes: Given two graphs $\mathcal{G}_1$ and $\mathcal{G}_2$ with set of constants Δ_1 and Δ_2 respectively, and the set of shapes $\mathcal{C}$, a *SHACL-homomorphism* μ from $\mathcal{G}$ to $\mathcal{G}'$ is a mapping $\mu : \Delta_1 \to \Delta_2$ such that, for each shape $s \in \mathcal{C}$ and each constant $v \in \Delta_1$, if $\langle \mathcal{G}_1, \mathcal{C} \rangle \models \phi_s(v)$ then $\langle \mathcal{G}_2, \mathcal{C} \rangle \models \phi_s(\mu(v))$.

Lemma 2 (canonical homomorphism for positive shapes). *Let $\mathcal{O}$ be a DL-Lite$_R$ ontology, $\mathcal{G}$ a graph, and let $\mathcal{M}$ be a minimal model of $\langle \mathcal{O}, \mathcal{G} \rangle$. Let $\mathcal{C}$ be a set of SHACL$^+$ shapes. Then, there is a SHACL-homomorphism from can$(\mathcal{O}, \mathcal{G})$ to $\mathcal{M}$ given $\mathcal{C}$. In particular, there exists a SHACL-homomorphism that maps every node from $\mathcal{G}$ to itself.*

Proof. From [9], we have that there exists a homomorphism μ from $can(\mathcal{O},\mathcal{G})$ to $\mathcal{M}$ such that for a basic concept C and node v it holds if $C(v) \in can(\mathcal{O},\mathcal{G})$ (resp. $R(v_1,v_2) \in can(\mathcal{O},\mathcal{G})$) then $C(\mu(v)) \in \mathcal{M}$ (resp. $R(\mu(v_1),\mu(v_2)) \in \mathcal{M}$. In particular, it is possible to select μ such that $\mu(v) = v$ for $v \in \mathcal{G}$. We also notice that μ has to be surjective; otherwise the μ-image of $can(\mathcal{O},\mathcal{G})$ would be a minimal model "smaller" than $\mathcal{M}$ which is a contradiction.

Assume now that $\langle can(\mathcal{O},\mathcal{G}),\mathcal{C}\rangle \models \phi_s(v)$ for some shape s from $\mathcal{C}$ and node v in $can(\mathcal{O},\mathcal{G})$. Let σ be a satisfying assignment for $can(\mathcal{O},\mathcal{G}),\mathcal{C}$ such that $[\![\phi_s]\!]^{can(\mathcal{O},\mathcal{G}),\sigma,v} = true$. We define an assignment σ' over $\mathcal{M},\mathcal{C}$ in the following way: for a shape s_1 and node v_1 in $\mathcal{M}$ we set $s_1 \in (v_1,\sigma')$ iff exists a node v_2 in $can(\mathcal{O},\mathcal{G})$ such that $\mu(v_2) = v_1$ and $s_1 \in (v_2,\sigma)$. Now analyzing different cases for s: $\phi_s = \top$, $\phi_s = I$, $\phi_s = \geq_k R.s_1$, $\phi_s = s_1 \wedge s_2$, $\phi_s = s_1 \vee s_2$ and $\phi_s = \mathrm{EQ}(r_1,r_2)$, it is not hard to show that if $[\![\phi_s]\!]^{can(\mathcal{O},\mathcal{G}),\sigma,v} = true$ then $[\![\phi_s]\!]^{\mathcal{M},\sigma',\mu(v)} = true$. □

Using the lemmas above we show the following property of canonical models.

Lemma 3 (canonical model characterization). *For a DL-Lite$_R$ ontology $\mathcal{O}$, basic concept C, graph G, node v in $\mathcal{G}$, set $\mathcal{C}$ of SHACL$^+$ shapes and shape s defined in $\mathcal{C}$ we have that: $\langle\mathcal{O},\mathcal{G},\mathcal{C}\rangle \models \phi_s(v)$ iff $\langle can(\mathcal{O},\mathcal{G}),\mathcal{C}\rangle \models \phi_s(v)$.*

Proof. (⇐) The entailment $\langle\mathcal{O},\mathcal{G},\mathcal{C}\rangle \models \phi_{s(v)}$ holds if $\langle\mathcal{M},\mathcal{C}\rangle \models \phi_{s(v)}$[7] in each minimal model $\mathcal{M}$, including $can(\mathcal{O},\mathcal{G})$.
(⇒) Let $\mathcal{M}$ be a minimal model of $\langle\mathcal{O},\mathcal{G}\rangle$. Lemma 2 implies the existence of a homomorphism μ from $can(\mathcal{O},\mathcal{G})$ to $\mathcal{M}$ s.t. $\mathcal{M},\mathcal{C} \models \phi_s(\mu(v))$ for $\mu(v) = v$. □

4.3 Rewriting of Shape Targets

In the absence of an ontology, the targets of shapes s are retrieved by evaluating the target definitions τ_s over the graph $\mathcal{G}$, written $[\![\tau_s]\!]^{\mathcal{G}}$. In SHACL, a target 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 Sect. 2.5, and retrieve targets over all minimal models, or equivalently over the canonical model, written as $[\![\tau]\!]^{\langle\mathcal{O},\mathcal{G}\rangle}$. To achieve this, since τ_s is a unary conjunctive query, one can apply PerfRef.

Lemma 4. *For any shape s from SHACL, DL-Lite$_R$ ontology $\mathcal{O}$, and graph $\mathcal{G}$ it holds that $[\![\tau_s]\!]^{\langle\mathcal{O},\mathcal{G}\rangle} = [\![\mathrm{PerfRef}(\tau_s,\mathcal{O})]\!]^{\mathcal{G}}$.*

Proof. The authors in [9] established the correspondence between certain answers of conjunctive queries over knowledge graphs and perfect reformulation (Lemma 35): For a KG $\langle\mathcal{O},\mathcal{G}\rangle$ and conjunctive query q we have that the certain answers of q over the KG correspond to perfect reformulation in the sense that $cert(q,\langle\mathcal{O},\mathcal{G}\rangle) = [\![\mathrm{PerfRef}(\tau_s,\mathcal{O})]\!]^{\mathcal{G}}$, where, $a \in cert(q,\langle\mathcal{O},\mathcal{G}\rangle)$ iff for every minimal model $\mathcal{M}$ of $\langle\mathcal{O},\mathcal{G}\rangle$ it holds that $a \in q^{\mathcal{M}}$. At the same time

[7] In this entailment we consider $\mathcal{M}$ as an infinite conjunction of atoms.

Algorithm 1. CONSTRAINT REWRITING

Input: ontology $\mathcal{O}$ possible with existential rules, set of positive shapes $\mathcal{C}$

1: $\mathcal{C}_\mathcal{O} \leftarrow \text{SHAPET}(\mathcal{O})$
2: $\mathcal{C}^v_\mathcal{O} \leftarrow \text{SHAPEVIRTUAL}(\mathcal{O})$
3: $\mathcal{C}^s_\mathcal{O} \leftarrow \text{SUCCESSORT}(\mathcal{O})$
4: $\mathcal{C}'' \leftarrow \{\langle \text{PERFREF}(\tau_s, \mathcal{O}), \text{REWRITECOMPL}(\phi_s, \mathcal{O})\rangle \mid s \in \mathcal{C}\}$
5: **return** $\mathcal{C}_\mathcal{O} \cup \mathcal{C}^v_\mathcal{O} \cup \mathcal{C}^s_{\mathcal{O},\mathcal{C}} \cup \mathcal{C}''$

the formula $[\![\tau_s]\!]^{\langle\mathcal{O},\mathcal{G}\rangle}$ defines a special set of target nodes over the graph and ontology: $[\![\tau_s]\!]^{\langle\mathcal{O},\mathcal{G}\rangle}$ returns a node v iff it does so for every minimal model M of $\langle\mathcal{O},\mathcal{G}\rangle$. □

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

4.4 Rewriting of the Ontology

In this part, we present our rewriting algorithm. In order to make notations more concise, we write $\mathcal{G}, \mathcal{C} \models \phi(v)$ to denote that the node v satisfies the constraint ϕ in the graph $\mathcal{G}$ given a set $\mathcal{C}$ of shapes. Similarly, we use $\langle\mathcal{O},\mathcal{G}\rangle, \mathcal{C} \models \phi(v)$ to denote that v satisfies ϕ in the graph that corresponds to the canonical model of $\langle\mathcal{O},\mathcal{G}\rangle$ given $\mathcal{C}$. Then, we assume that the shape constraints in $\mathcal{C}$ are normalised, i.e. contain at most one operator. Note that this can always be obtained by introducing nested shape names.

Our rewriting procedure is presented in Algorithm 1 and we now guide the reader through it. Our algorithm relies on auxiliary shapes of three kinds:

- $\mathcal{C}_\mathcal{O}$ that contains for each concept C in $\mathcal{O}$ the corresponding shape s_C in $\mathcal{C}_\mathcal{O}$; this ensures that for every node v the fact $C(v)$ is in the canonical model iff v is valid in the shape s_C.
- "virtual" shapes $\mathcal{C}^v_\mathcal{O}$ and $\mathcal{C}^s_\mathcal{O}$ that are used to capture the part of the canonical model generated by existential quantification.

The shapes $\mathcal{C}_\mathcal{O}$, $\mathcal{C}^v_\mathcal{O}$ and $\mathcal{C}^s_\mathcal{O}$ will help us to establish rewriting over $\mathcal{O}$ of the original shapes from $\mathcal{C}$ into $\mathcal{C}''$. We now define $\mathcal{C}_\mathcal{O}$, $\mathcal{C}^v_\mathcal{O}$ and $\mathcal{C}^s_\mathcal{O}$ explain how they are used in Algorithm 1 and show correctness of the algorithm.

Rewriting SHAPET for Active Nodes: $\mathcal{C}_\mathcal{O}$. For every concept of the form A (resp. $\exists R$) in $\mathcal{O}$, we introduce a shape s_A (resp. $s_{\exists R}$), with targets $\tau_{s_A} = A$ (resp. $\tau_{s_{\exists R}} = \exists R$) and with constraint where R, R' may be inverse roles:

$$\phi_{s_A} = (\geq_1 \mathsf{a}.A) \vee \bigvee_{\mathcal{O} \models C \sqsubseteq A} s_C, \qquad \phi_{s_{\exists R}} = (\geq_1 R.\top) \vee \bigvee_{\mathcal{O} \models C \sqsubseteq \exists R} s_C \vee \bigvee_{\mathcal{O} \models R' \sqsubseteq R} s_{\exists R'}.$$

Next, we introduce shapes that encode negative assertions. For each GCI of the form $(C \sqsubseteq \neg D)$ in $\mathcal{O}$, we introduce one shape $s_{C \sqsubseteq \neg D}$, where targets are all instances of C and D in $\mathcal{G}$, where the constraint is always violated. To this end,

we exploit results based on PerfRef [9]: $\tau_{s_{C \sqsubseteq \neg D}} = \text{PerfRef}(C(x) \wedge D(x), \mathcal{O})$, and $\phi_{s_{C \sqsubseteq \neg D}} = \bot$. Similarly, for negative role inclusions, we use $s_{R_1 \sqsubseteq \neg R_2}$, with $\tau_{s_{R_1 \sqsubseteq \neg R_2}} = \text{PerfRef}(\exists y R1(x, y) \wedge R_2(x, y), \mathcal{O})$, and $\phi_{s_{R_1 \sqsubseteq \neg R_2}} = \bot$.

We denote the set of shapes produced above with $\mathcal{C}_\mathcal{O}$, and the corresponding translation function as ShapeT in Algorithm 1, i.e., $\text{ShapeT}(\mathcal{O}) = \mathcal{C}_\mathcal{O}$.

With $cl(\mathcal{O}, \mathcal{G})$ we denote the maximal subset of $can(\mathcal{O}, \mathcal{G})$ without fresh nodes. If there were no fresh nodes, i.e., when $can(\mathcal{O}, \mathcal{G}) = cl(\mathcal{O}, \mathcal{G})$, we observe that shapes in $\mathcal{C} \cup \text{ShapeT}(\mathcal{O})$ would be sufficient to validate the facts in $can(\mathcal{O}, \mathcal{G})$. Intuitively, this is because dependencies among shapes in $\text{ShapeT}(\mathcal{O})$ corresponds to the construction of the "closure" $cl(\mathcal{O}, \mathcal{G})$.

Fresh Nodes in Canonical Models. Observe that for DL-Lite$_R$ ontologies $\mathcal{O}$ the canonical model $can(\mathcal{O}, \mathcal{G})$ can be arbitrary large (even infinite) in which case it will contain "fresh" nodes that are not occurring in $\mathcal{G}$. Thus, one should be able to check constraints also on this fresh nodes, as shown next.

Example 7. Consider the ontology $\mathcal{O} = \{A \sqsubseteq \exists U, \exists U^- \sqsubseteq \exists P\}$ and graph $\mathcal{G} = \{(v, \mathsf{a}, A)\}$. Then $can(\mathcal{O}, \mathcal{G}) = \{(v, \mathsf{a}, A), (v, U, a_1), (a_1, P, a_2)\}$ 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 $\mathcal{C}$ is valid over $(\mathcal{O}, \mathcal{G})$. □

These properties of $can(\mathcal{O}, \mathcal{G})$ make rewriting technically involved since SHACL constraints cannot express fresh values. We address this with auxiliary shapes $\mathcal{C}^v_\mathcal{O}$ and $\mathcal{C}^s_\mathcal{O}$ that mimic the construction of the canonical model and validate facts in $\mathcal{G}' = can(\mathcal{O}, \mathcal{G}) \setminus cl(\mathcal{O}, \mathcal{G})$. The graph $\mathcal{G}'$ is a forest (by construction of $can(\mathcal{O}, \mathcal{G})$) where each tree has the root in some assertion of $cl(\mathcal{O}, \mathcal{G})$. We call this root the *witness* of the tree. In example above, (v, a, A) is the only witness.

Rewriting shapeVirtual for Fresh Nodes: $\mathcal{C}^v_\mathcal{O}$. For each concept C appearing in a GCI in $\mathcal{O}$, we introduce a shape $s^{virtual}_C$, such that, for a node v in $\mathcal{G}$, verifies $s^{virtual}_C(v)$ iff there is a node v' in $\mathcal{G}'$ with v as witness such that $\mathcal{G}' \models C(v')$. For instance, in Example 8, we introduce shape $s^{virtual}_{\exists R}$ which is verified by the witness v. Note that v' is not necessarily an immediate successor of v in $\mathcal{G}'$.

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

Rewriting successorT for Fresh Nodes: $\mathcal{C}^s_\mathcal{O}$. The second kind of shapes is needed in order to check if two roles are concatenated in the same tree in $\mathcal{G}'$. For each pair of roles R_1 and R_2 in $\mathcal{O}$, we introduce the shape $s^{succ}_{R_1,R_2}$ such that a node $v \in \mathcal{G}$ verifies $\phi_{s^{succ}_{R_1,R_2}}$ iff (a_1, R_1, a_2) and (a_2, R_2, a_3) are on the subtree with

the witness v, for some a_1, a_2, a_3 in $\mathcal{G}'$. In our running example, v verifies $\phi_{s^{succ}_{R,P}}$, but not $\phi_{s^{succ}_{P,R}}$. Formally, for every two roles R_1 and R_2 in $\mathcal{O}$, $\tau_{s^{succ}_{R_1,R_2}} = \bot(x)$, and if $\mathcal{O} \models \exists R_1^- \sqsubseteq \exists R_2$, then $\phi_{s^{succ}_{R_1,R_2}} = s_{\exists R_1}$, otherwise $\phi_{s^{succ}_{\exists R_1,\exists R_2}} = \bot$. The special case $R_2 = R_1^-$, is also covered by the definition $\phi_{s^{succ}_{R_1,R_1^-}} = s_{\exists R_1}$. Let SUCCESSORT denote the function creating these fresh shapes.

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

$$\phi'_s = (\geq_k R.s'_1) \vee s^{virtual} \quad \text{where} \quad \phi_{s^{virtual}} = s^{virtual}_{\exists R} \wedge s^{virtual}_1 \wedge s_{\exists R,s_1}.$$

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

Correctness of Rewriting. We now proceed to the correctness of our rewriting procedure and start with the property of possibly infinite canonical models.

Lemma 5 (Infinite canonical model). *Let $\mathcal{O}$ be a DL-Lite$_R$ ontology, C a concept in $\mathcal{O}$, R and P properties in $\mathcal{O}$, $\mathcal{G}$ a graph, v a node in $\mathcal{G}$, and shapes $\mathcal{C}_\mathcal{O}$, $\mathcal{C}^v_\mathcal{O}$ and $\mathcal{C}^s_\mathcal{O}$ as specified in Algorithm 1. Then, the following holds:*

- $\mathcal{G}, \mathcal{C}_\mathcal{O} \cup \mathcal{C}^v_\mathcal{O} \cup \mathcal{C}^s_\mathcal{O} \models \phi_{s^{virtual}_C}(v)$ *iff there is a node a_1 in $can(\mathcal{O},\mathcal{G})$ with the witness v such that $can(\mathcal{O},\mathcal{G}) \models C(a_1)$.*
- $\mathcal{G}, \mathcal{C}_\mathcal{O} \cup \mathcal{C}^v_\mathcal{O} \cup \mathcal{C}^s_\mathcal{O} \models \phi_{s^{\text{succ}}_{R,P}(v)}$ *iff there are nodes a_1, a_2, a_3 in $can(\mathcal{O},\mathcal{G})$ with the witness v such that $can(\mathcal{O},\mathcal{G}) \models R(a_1,a_2)$ and $can(\mathcal{O},\mathcal{G}) \models P(a_2,a_3)$.*

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

$$\phi_{s^{virtual}_A} = \underline{s_A}, \quad \phi_{s^{virtual}_{\exists U}} = s_{\exists U} \vee \underline{s^{virtual}_A} \vee s^{virtual}_{\exists U^-},$$
$$\phi_{s_{\exists U}} = (\geq_1 U.\top) \vee \underline{s^{virtual}_{\exists U}}, \quad \phi_{s^{succ}_{U,P}} = \underline{s_{\exists U}}, \quad \phi_{s_{\exists U,s_2}} = \underline{s^{succ}_{U,P}},$$
$$\phi_{s^{virtual}_1} = \underline{s^{virtual}_{\exists U}} \wedge \underline{s^{virtual}_2} \wedge \underline{s_{\exists U,s_2}}, \quad \phi_{s'_1} = \geq_1 U.s'_2 \vee \underline{s^{virtual}_1},$$
$$\phi_{s^{virtual}_2} = \underline{s^{virtual}_{\exists P}}, \quad \phi_{s^{virtual}_{\exists P}} = s_{\exists P} \vee \underline{s^{virtual}_{\exists U^-}} \vee s^{virtual}_{\exists P^-}, \quad \phi_{s^{virtual}_{\exists U^-}} = \underline{s^{virtual}_{\exists U}}.$$

The only target of s is v, and v verifies ϕ_s w.r.t the rewritten set of shapes. □

We are ready to present the main result of this section.

Theorem 2. *Let $\mathcal{O}$ be a DL-Lite$_R$ ontology, $\mathcal{C}$ a set of SHACL$^+$ shapes, $s \in \mathcal{C}$, $\mathcal{C}'$ the shapes returned by Algorithm 1, and s' the rewriting of s in $\mathcal{C}'$. Then, for any graph $\mathcal{G}$ and node v in $\mathcal{G}$ it holds that: $\langle \mathcal{O}, \mathcal{G} \rangle, \mathcal{C} \models \phi_s(v)$ iff $\mathcal{G}, \mathcal{C}' \models \phi'_s(v)$.*

Note the size of the returned rewriting is polynomial in the size of $\mathcal{O}$ and $\mathcal{C}$.

5 Related Work

As discussed in Sect. 1, constraint validation in the presence of ontologies was studied in [20,35,42]. While these approaches allow for very expressive ontologies (e.g., $\mathcal{SHOIN}$) they require an engine for disjunctive logic programs, which we believe makes these approaches less practically interesting.

Rewriting of conjunctive queries over OWL 2 QL ontologies of [9] cannot be applied to SHACL shapes since they may be recursive or contain negation.

From a more general prospective our rewriting can be seen a special case of the *backward-chaining algorithm* [32] over ∀∃-rules (where the body and the head are conjunctions of atoms, and only the variables that occur in the head can be existentially quantified). However, in such cases the existence of a rewriting is undecidable in general for arbitrary rules, and even for some decidable fragments, differently from our approach, such algorithms may not terminate.

Naturally, one may think of relating SHACL to Datalog programs [14] since they poses recursion and negation. However, Datalog programs can at most have one unique minimal model, and SHACL constraints should be checked for all possible assignments [12] (including also non-minimal ones). If we consider more expressive version of Datalog like Datalog with negation under the stable model semantics (SMS) [14], then relating it to SHACL is more promising, while the actual relation is not obvious as SMS is also based on minimal models. Nevertheless, our preliminary results show that this is not straightforward.

6 Conclusion

We have studied the rewriting of constraints over ontologies for constrain validation. We focused on a prominent language for graph constraints 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 on SHACL to positive (but recursive) fragment SHACL$^+$ for which they always exist. For SHACL$^+$ we showed how to rewrite ontologies and SHACL$^+$ constraints into the unique set of SHACL$^+$ constraints. Moreover, SHACL$^+$ validation over OWL 2 QL is tractable.

We see this work as an important step towards practical constraint rewriting algorithms and systems. Next, we plan to analyse optimisation 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., [38]).

Then, we plan to extend this work to account for OWL 2 EL. Moreover, we plan to implement our approach and evaluate it in various industrial settings [11,24–27,33,34,37,39]. An important research direction is also to understand how to repair data that fails to satisfy SHACL constraints and we see the work on ontology evolution as a good source of inspiration [10,29,30,43].

Acknowledgments. This work was partially funded by the SIRIUS Centre, Norwegian Research Council project number 237898; by the Free University of Bozen-Bolzano projects QUEST, ROBAST and ADVANCED4KG.

References

1. Freebase: an open, shared database of the world's knowledge. freebase.com/
2. Google KG. google.co.uk/insidesearch/features/search/knowledge.html
3. W3C: OWL 2 Web Ontology Language. www.w3.org/TR/owl2-overview/
4. Abiteboul, S., Buneman, P., Suciu, D.: Data on the Web: From Relations to Semistructured Data and XML. Morgan Kaufmann, Burlington (1999)
5. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Boston (1995)
6. Arenas, M., Grau, B.C., Kharlamov, E., Marciuska, S., Zheleznyakov, D.: Faceted search over RDF-based knowledge graphs. J. Web Semant. **37–38**, 55–74 (2016)
7. Arenas, M., Gutiérrez, C., Pérez, J.: Foundations of RDF databases. In: Tessaris, S., et al. (eds.) Reasoning Web 2009. LNCS, vol. 5689, pp. 158–204. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03754-2_4
8. Boneva, I., Labra Gayo, J.E., Prud'hommeaux, E.G.: Semantics and validation of shapes schemas for RDF. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10587, pp. 104–120. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68288-4_7
9. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: the DL-Lite family. JAR **39**, 385–429 (2007)
10. Calvanese, D., Kharlamov, E., Nutt, W., Zheleznyakov, D.: Evolution of *DL – Lite* knowledge bases. In: Patel-Schneider, P.F., et al. (eds.) ISWC 2010. LNCS, vol. 6496, pp. 112–128. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-17746-0_8
11. Cheng, G., Kharlamov, E.: Towards a semantic keyword search over industrial knowledge graphs (extended abstract). In: IEEE Big Data, pp. 1698–1700 (2017)
12. Corman, J., Reutter, J.L., Savković, O.: Semantics and validation of recursive SHACL. In: Vrandečić, D., et al. (eds.) ISWC 2018. LNCS, vol. 11136, pp. 318–336. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-00671-6_19
13. Corman, J., Reutter, J.L., Savkovic, O.: Semantics and validation of recursive SHACL (extended version). Technical report KRDB18-1, KRDB Research Center, Free University of Bozen-Bolzano (2018). https://www.inf.unibz.it/krdb/pub/tech-rep.php
14. Dantsin, E., Eiter, T., Gottlob, G., Voronkov, A.: Complexity and expressive power of logic programming. ACM Comput. Surv. **33**(3), 374–425 (2001)
15. Ekaputra, F.J., Lin, X.: SHACL4p: SHACL constraints validation within Protégé ontology editor. In: ICoDSE (2016)

16. Fan, W., Fan, Z., Tian, C., Dong, X.L.: Keys for graphs. PVLDB **8**(12), 1590–1601 (2015)
17. Fan, W., Wu, Y., Xu, J.: Functional dependencies for graphs. In: SIGMOD, pp. 1843–1857 (2016)
18. Hansen, P., Lutz, C., Seylan, I., Wolter, F.: Efficient query rewriting in the description logic EL and beyond. In: IJCAI, pp. 3034–3040 (2015)
19. Horrocks, I., Giese, M., Kharlamov, E., Waaler, A.: Using semantic technology to tame the data variety challenge. IEEE Internet Comput. **20**(6), 62–66 (2016)
20. Kharlamov, E., et al.: Capturing industrial information models with ontologies and constraints. In: Groth, P., et al. (eds.) ISWC 2016. LNCS, vol. 9982, pp. 325–343. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46547-0_30
21. Kharlamov, E., et al.: SOMM: industry oriented ontology management tool. In: ISWC Posters & Demos (2016)
22. Kharlamov, E., et al.: Ontology based data access in Statoil. J. Web Semant. **44**, 3–36 (2017)
23. Kharlamov, E., et al.: Semantic access to streaming and static data at Siemens. J. Web Semant. **44**, 54–74 (2017)
24. Kharlamov, E., Martín-Recuerda, F., Perry, B., Cameron, D., Fjellheim, R., Waaler, A.: Towards semantically enhanced digital twins. In: IEEE Big Data, pp. 4189–4193 (2018)
25. Kharlamov, E., et al.: Towards simplification of analytical workflows with semantics at Siemens (extended abstract). In: IEEE Big Data, pp. 1951–1954 (2018)
26. Kharlamov, E., et al.: Diagnostics of trains with semantic diagnostics rules. In: Riguzzi, F., Bellodi, E., Zese, R. (eds.) ILP 2018. LNCS (LNAI), vol. 11105, pp. 54–71. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-99960-9_4
27. Kharlamov, E., et al.: Semantic rules for machine diagnostics: execution and management. In: CIKM, pp. 2131–2134 (2017)
28. Kharlamov, E., et al.: Finding data should be easier than finding oil. In: IEEE Big Data, pp. 1747–1756 (2018)
29. Kharlamov, E., Zheleznyakov, D.: Capturing instance level ontology evolution for DL-Lite. In: Aroyo, L., et al. (eds.) ISWC 2011. LNCS, vol. 7031, pp. 321–337. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-25073-6_21
30. Kharlamov, E., Zheleznyakov, D., Calvanese, D.: Capturing model-based ontology evolution at the instance level: the case of DL-Lite. J. Comput. Syst. Sci. **79**(6), 835–872 (2013)
31. Knublauch, H., Ryman, A.: Shapes constraint language (SHACL). W3C Recommendation, vol. 11, no. 8 (2017)
32. König, M., Leclère, M., Mugnier, M., Thomazo, M.: Sound, complete and minimal UCQ-rewriting for existential rules. Semant. Web **6**(5), 451–475 (2015)
33. Mehdi, G., et al.: Semantic rule-based equipment diagnostics. In: d'Amato, C., et al. (eds.) ISWC 2017. LNCS, vol. 10588, pp. 314–333. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-68204-4_29
34. Mehdi, G., et al.: SemDia: semantic rule-based equipment diagnostics tool. In: CIKM, pp. 2507–2510 (2017)
35. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. Web Semant. Sci. Serv. Agents World Wide Web **7**(2), 74–89 (2009)
36. Patel-Schneider, P.F.: Using description logics for RDF constraint checking and closed-world recognition. In: AAAI (2015)
37. Ringsquandl, M., et al.: On event-driven knowledge graph completion in digital factories. In: IEEE Big Data, pp. 1676–1681 (2017)

38. Savkovic, O., Calvanese, D.: Introducing datatypes in DL-Lite. In: ECAI, pp. 720–725 (2012)
39. Savković, O., et al.: Semantic diagnostics of smart factories. In: Ichise, R., Lecue, F., Kawamura, T., Zhao, D., Muggleton, S., Kozaki, K. (eds.) JIST 2018. LNCS, vol. 11341, pp. 277–294. Springer, Cham (2018). https://doi.org/10.1007/978-3-030-04284-4_19
40. Soylu, A., et al.: OptiqueVQS: a visual query system over ontologies for industry. Semant. Web **9**(5), 627–660 (2018)
41. Suchanek, F.M., Kasneci, G., Weikum, G.: YAGO: a core of semantic knowledge. In: Proceedings of WWW, pp. 697–706 (2007)
42. Tao, J., Sirin, E., Bao, J., McGuinness, D.L.: Integrity constraints in OWL. In: AAAI (2010)
43. Zheleznyakov, D., Kharlamov, E., Horrocks, I.: Trust-sensitive evolution of DL-Lite knowledge bases. In: AAAI, pp. 1266–1273 (2017)