Optimizing SPARQL Queries with SHACL

Ratan Bahadur Thapa and Martin Giese[0000-0002-2058-2728]

Department of Informatics, University of Oslo, Oslo, Norway
{ratanbt|martingi}@ifi.uio.no

Abstract. We propose a set of optimizations that can be applied to a given SPARQL query, and that guarantee that the optimized query has the same answers under bag semantics as the original query, provided that the queried RDF graph validates certain SHACL constraints. Our optimizations exploit the relationship between graph patterns in the SPARQL queries and the SHACL constraints that describe those patterns in the RDF graph. We prove the correctness of these optimizations and show how they can be propagated to larger queries while preserving answers. Further, we prove the confluence of rewritings that employ these optimizations, guaranteeing convergence to the same optimized query regardless of the rewriting order.

1 Introduction

The Resource Description Framework (RDF) [37] has seen increasing adoption in recent years, prompting the W3C standardization of the SPARQL [46,24] query language for RDF. Since the W3C standardization in 2008, SPARQL has been recognised as a key technology for the Semantic Web and the current version SPARQL 1.1 [24] is well-adopted in both academia and industries, e.g., query engines for AllegroGraph [22], Apache Jena [16], Sesame [11] and OpenLink Virtuoso [21].

The RDF model, as an abstract knowledge representation, doesn't explicitly distinguish between data and metadata, such as schema and constraints. To address this, the W3C has recommended the SHACL [32] constraint language for RDF. SHACL validation relies on "shapes," which define a set of constraints and indicate which nodes in an RDF graph should be validated against these constraints. Since its recommendation in 2017, SHACL has been widely used to verify compliance of RDF datasets w.r.t. certain policies, such as GDPR requirements [39], and to generate guarantees for data transformation [53,54], facilitating data repairs [4] and others [40]. Additionally, a set of SHACL constraints can function as a "schema" for RDF datasets that satisfy these constraints, enhancing the understandability and usability of the data represented in RDF.

However, the potential of utilizing SHACL to optimize SPARQL queries, similar to how relational constraints optimize queries in relational databases [3], remains largely unexplored. While hand-written SPARQL queries are often tailored to the structure of the RDF graph being queried, queries generated by user interfaces [31,38] or query translators [28,57] and rewriting engines [26,52,36] may contain redundant parts that are not necessary for a specific graph. SHACL can capture information about the structure of an RDF graph, similar to relational database constraints. As a result, triple stores that have built-in support for SHACL can then use this information to enforce SHACL constraints as well as incorporate them into query processing to optimize queries. To address this issue, we propose a set of optimizations that maintain the correctness of query results and can be applied to a SPARQL query prior to processing, provided that the RDF graph conforms to a specified set of SHACL shapes.

SPARQL queries consist of three main components: (a) pattern matching, which includes optional parts, unions, joins, nesting, and filtering values to find possible matches, (b) solution modifiers, which allow for modifying matched values using operators such as projection and distinct, and (c) output mode, which can be a boolean answer or selections of values from variables that match the patterns, constructing new values, or describing existing data through constraints. To optimize SPARQL queries, we focus on those containing semantically redundant graph patterns that can be minimized or restructured in the query definition based on SHACL descriptions of these patterns in RDF.

The optional operator is a crucial feature in SPARQL queries, commonly used to handle missing or unavailable information [6]. By using optional, query answers can include available information instead of failing to give an answer when parts of the pattern do not match. This feature is particularly important in semantic web applications, where partial knowledge about data is often assumed. However, optional is also a source of complexity in query answering, as it is PSPACE-hard for optional alone [43,50]. As a result, it has been actively studied [34,29] and optimized [56,15] in various query circumstances. In this work, we focus on the Left-Join intuition applied by Xiao et al. [56] for the treatment of optional. However, instead of relying on SQL not null constraint information in their SPARQL-to-SQL translation, we use SHACL constraints to reduce optional patterns to joins. We illustrate this approach using an example below.

Example 1 *Consider an RDF graph on the left that validates a SHACL shape s on the right, written in Turtle syntax:*

```
:Ida a :Student;
  :hasID "001"^^xsd:int;
  :hasAddress "Oslo".
:Ingrid a :Student;
  :hasID "002"^^xsd:int;
  :hasAddress "Bergen".
```

```
:StudentNode a sh:NodeShape;
 sh:targetClass :Student;
 sh:property [ sh:path :hasAddress;
             sh:nodeKind sh:Literal;
             sh:maxCount 1;  sh:minCount 1 ; ];
 sh:property [ sh:path :hasID;
              dash:uniqueValueForClass
                       :Student ; ].
```

The node shape, '`:StudentNode`' declares (a) a class-based target, indicating that all the members of `Student` class are target nodes, (b) cardinality path constraint requiring all students to have exactly one address, and (c) `dash:uniqueValueForClass`[1] path constraint stating that no two students can share the same id. Instances of `Student` validate against the shape if they satisfy both stated constraints.

Consider an information need to retrieve the `ids` of students including their addresses if available, which could be expressed as the following SPARQL query Q,

SELECT ?y ?z WHERE {?x a Student . ?x :hasID ?y . Optional {?x :hasAddress ?z}} .

[1] https://datashapes.org/constraints.html

Q retrieves `ids` of students including their `addresses` if possible from the graph, i.e., in the absence of `address`, `ids` are still retrieved with variable $?z$ left undefined in the answer. Observe a scenario, where Q is to be evaluated over a graph that satisfies the shape s. Since the shape s guarantees that for each student $?x$, irrespective of their ids $?y$, variable $?z$ will always bind to an address (i.e., will never be left unbound), the query Q can be rewritten as follows without the `Optional`,

$$\texttt{SELECT}\,?y\;?z\;\texttt{WHERE}\,\{?x\;a\;\texttt{Student}\,.\,?x\;\texttt{:hasID}\;?y\,.\,?x\;\texttt{:hasAddress}\;?z\,.\}$$

Similarly, consider finding the `ids` of those students who have addresses, such as,

$$\texttt{SELECT}\,?y\;\texttt{WHERE}\,\{?x\;a\;\texttt{Student}\,.\,?x\;\texttt{:hasID}\;?y\,.\,?x\;\texttt{:hasAddress}\;?z\,.\}$$

Since shape s guarantees an address $?z$ for every student $?x$, the original query can be rewritten as follows without changing the solutions,

$$\texttt{SELECT}\,?y\;\texttt{WHERE}\,\{?x\;a\;\texttt{Student}\,.\,?x\;\texttt{:hasID}\;?y\,.\}$$

Consider the query `SELECT DISTINCT` $?y$ `WHERE` $\{?x\;a$ `Student` $.\;?x$ `:hasID` $?y\,.\}$. Then, the distinct operator can be removed from the query definition since the shape s guarantees a unique id $?y$ for every student $?x$.

Our approach to optimizing joins in SPARQL is similar to that of query containment concerning inclusion dependencies [27] in relational databases. In particular, when dealing with two query patterns $\mathcal{P}_1$ and $\mathcal{P}_2$ under set (resp., bag) semantics, we can simplify their join by reducing it to $\mathcal{P}_2$ if the answers to $\mathcal{P}_1$ are included in the answers to $\mathcal{P}_2$ for any graph. To achieve this, we draw on intuitions from previous literature [49,33,13,51,9] on join optimization. Specifically, we focus on the task of pruning join patterns that do not affect the output of a query, resulting in reduced query execution costs. Finally, eliminating distinct patterns can also improve query performance significantly as it can be a bottleneck in SPARQL processing, requiring substantial resources to eliminate duplicate rows, particularly with large datasets [7]. We remove distinct if the original query's result is guaranteed to be free of duplicate rows by SHACL. For instance, when joining two triple patterns with a unique one-to-one (resp., one to at most one) relation, applying distinct over the join may not have any effect. An example of this optimization is illustrated in Example 1.

The rest of the paper is organized as follows: In Section 2, we review fundamental concepts of SPARQL algebra and SHACL constraints. Query optimizations are presented in Section 3. Section 4 contains a discussion of our results, and Section 5 concludes the paper. Due to space constraints, the paper includes only partial proofs of the results. For details proofs, we refer the reader to [55].

2 Preliminaries

We next recapitulate the SPARQL algebra and SHACL constraints that we deal with in this paper.

RDF Graph. Assume that $\mathbf{I}$, $\mathbf{B}$ and $\mathbf{L}$ are countably infinite disjoint sets of *Internationalized Resource Identifiers* (IRIs), *Blank nodes* and *Literals*, respectively. The set of RDF terms $\mathbf{T}$ is $\mathbf{I} \cup \mathbf{B} \cup \mathbf{L}$. An *RDF triple* is an element $\langle s, p, o\rangle$ of $(\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times \mathbf{T}$, where s is called the subject, p the predicate and o the object. An RDF graph $G \subseteq (\mathbf{I} \cup \mathbf{B}) \times \mathbf{I} \times \mathbf{T}$ is a finite set of RDF triples. We assume that all RDF graphs are non-empty, which we do not see as a limitation in practice since querying empty graphs has no use in most applications. For simplicity, we represent RDF triples as binary relations in first-order logic, except the triple of the form $\langle s, \text{rdf:type}, C\rangle$. We write $P(s, o)$ (resp., $C(s)$) for the RDF triple $\langle s, P, o\rangle$ (resp., $\langle s, \text{rdf:type}, C\rangle$) and $P^-(s, o)$ for the inverse of triple $P(o, s)$.

Definition 1 *The set $\mathcal{N}_G$ of nodes of an RDF graph G is the union of sets of subjects and objects of triples of the form $\langle s, P, o\rangle$ and subjects of triples of the form $\langle s, \mathit{rdf{:}type}, C\rangle$ in the graph, i.e., $\{s, o \mid P(s, o) \in G \text{ or } C(s) \in G\}$.*

SPARQL Algebra. We formally adopt the bag-based (rather than the set-based [43]) semantics for SPARQL as in the W3C specification [46,24] and studied in the literature [5,30].

Assume a countably infinite set $\mathbf{V}$ of variables disjoint from $\mathbf{T}$. A triple pattern is defined as a triple in $(\mathbf{I} \cup \mathbf{L} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{V}) \times (\mathbf{I} \cup \mathbf{L} \cup \mathbf{V})$. We use the same notations $P(x, y)$, $P^-(x, y)$, $C(x)$ for triple patterns as we do for RDF triples. A *basic graph pattern (BGP)* is a finite set of triple patterns. For simplicity, we only consider project and distinct solution modifiers and treat SPARQL query as a graph pattern $\mathcal{P}$, defined by the grammar

$$\mathcal{P} ::= \mathcal{B} \mid \textsc{Filter}_{\mathcal{F}}(\mathcal{P}) \mid \textsc{Union}(\mathcal{P}_1, \mathcal{P}_2) \mid \textsc{Join}(\mathcal{P}_1, \mathcal{P}_2) \mid \textsc{Minus}(\mathcal{P}_1, \mathcal{P}_2) \mid \\ \textsc{Diff}_{\mathcal{F}}(\mathcal{P}_1, \mathcal{P}_2) \mid \textsc{Opt}_{\mathcal{F}}(\mathcal{P}_1, \mathcal{P}_2) \mid \textsc{Proj}_L(\mathcal{P}) \mid \textsc{Dist}(\mathcal{P}),$$

where $\mathcal{B}$ is a BGP, $L \subseteq \mathbf{V}$ and $\mathcal{F}$, called *filter*, is a formula constructed using the logical connectives $\wedge$ and $\neg$ from atoms of the form bound(v), $(v = c)$, $(v = v')$ for $v, v' \in \mathbf{V}$ and $c \in \mathbf{T}$. Let $var(\mathcal{P})$ (resp., $var(\mathcal{F})$) be the set of variables occurring in a graph pattern $\mathcal{P}$ (resp. filter $\mathcal{F}$). In the rest of the article, we assume that all filter constraints are safe, i.e., $var(\mathcal{F}) \subseteq var(\mathcal{P})$ for every pattern $\textsc{Filter}_{\mathcal{F}}(\mathcal{P})$.

Given a SPARQL query Q and graph pattern $\mathcal{P}$, we write $\mathcal{P} \trianglelefteq Q$ if either $\mathcal{P}$ is Q or $\mathcal{P}$ appears in Q. Similarly, we denote by $Q^{\mathcal{P} \mapsto \mathcal{P}'}$ the rewriting of $\mathcal{P} \trianglelefteq Q$ to graph pattern $\mathcal{P}'$ in Q, $Q^{X \mapsto Y}$ the renaming of $X \subseteq var(Q)$ to a set Y of variables in Q (i.e., renaming $\rho : X \to Y$ s.t. $\forall y \in Y, \exists x \in X$ and $\rho(x) = y$), $var(Q \setminus \mathcal{P})$ the set of variables that occur in Q excluding the pattern $\mathcal{P} \trianglelefteq Q$ and $var(Q \cup \mathcal{P}) := var(Q) \cup var(\mathcal{P})$. For readability, we also write nested $\textsc{Join}$ expressions as *concatenation* whenever required, e.g., the graph pattern $\mathcal{P} = \textsc{Join}(\ldots \textsc{Join}(\mathcal{P}_1, \mathcal{P}_2), \ldots \mathcal{P}_n)$ may be written as $\mathcal{P}_1\mathcal{P}_2 \ldots \mathcal{P}_n$, and the $\mathcal{P}_i$ are called inner graph patterns of $\mathcal{P}$.

The semantics of graph patterns is defined in terms of (solution) mappings, partial functions, $\mu : \mathbf{V} \to \mathbf{T}$ with (possibly empty) domain dom$_\mu$. Let $\mu_{|L}$ (resp., $\mu_{|\bar{L}}$) be the restriction of mapping μ to $L \subseteq \mathbf{V}$ (resp. $\mathbf{V} \setminus L$) . Two mappings μ_1 and μ_2 are called compatible, denoted by $\mu_1 \sim \mu_2$, if $\mu_1(v) = \mu_2(v)$, for all $v \in \text{dom}_{\mu_1} \cap \text{dom}_{\mu_2}$, in which case $\mu_1 \oplus \mu_2$ denotes a solution mapping with domain $\text{dom}_{\mu_1} \cup \text{dom}_{\mu_2}$ s.t. $\mu_1 \oplus \mu_2 : v \to \mu_1(v)$ for $v \in \text{dom}_{\mu_1}$, and $\mu_1 \oplus \mu_2 : v \to \mu_2(v)$ for $v \in \text{dom}_{\mu_2}$. The truth-value $\mathcal{F}^\mu \in \{\top, \bot, \varepsilon\}$ (i.e., ε stands for "error") of a filter $\mathcal{F}$ under a mapping μ is defined inductively,

- □ $(\text{bound}(v))^\mu = \begin{cases} \top, & \text{if } v \in \text{dom}_\mu, \\ \bot, & \text{otherwise.} \end{cases}$
- □ $(v = c)^\mu = \begin{cases} \top, & \text{if } \mu(v) = c, \\ \bot, & \text{if } \mu(v) \neq c, \\ \varepsilon \text{ (error)}, & \text{if } v \notin \text{dom}_\mu, \end{cases}$ and $(v = v')^\mu = \begin{cases} \top, \text{if } \mu(v) = \mu(v'), \\ \bot, \text{if } \mu(v) \neq \mu(v'), \\ \varepsilon, \text{if } \{v, v'\} \nsubseteq \text{dom}_\mu \,. \end{cases}$
- □ $(\neg\mathcal{F})^\mu = \begin{cases} \top, & \text{if } \mathcal{F}^\mu = \bot, \\ \bot, & \text{if } \mathcal{F}^\mu = \top, \\ \varepsilon, & \text{if } \mathcal{F}^\mu = \varepsilon, \end{cases}$ and $(\mathcal{F}_1 \wedge \mathcal{F}_2)^\mu = \begin{cases} \top, \text{if } {\mathcal{F}_1}^\mu = {\mathcal{F}_2}^\mu = \top, \\ \bot, \text{if } {\mathcal{F}_1}^\mu = \bot \text{ or } {\mathcal{F}_2}^\mu = \bot, \\ \varepsilon, \text{if } {\mathcal{F}_1}^\mu = \varepsilon \text{ or } {\mathcal{F}_2}^\mu = \varepsilon \,. \end{cases}$

The evaluation of a SPARQL graph pattern $\mathcal{P}$ over an RDF graph G, denoted by $\mathcal{P}^G$, returns a multiset (i.e., bag) of mappings. Let $|\mu, \mathcal{P}^G|$ be the multiplicity of a mapping μ in the multiset $\mathcal{P}^G$. In this sense, we also write $\mu \in \mathcal{P}^G$ when $|\mu, \mathcal{P}^G| > 0$ and $\mu \notin \mathcal{P}^G$ when $|\mu, \mathcal{P}^G| = 0$. Then, the $|\mu, \mathcal{P}^G|$ is defined recursively as follows:

1. Let $\mathcal{P}$ be a BGP. Then, $|\mu, \mathcal{P}^G| = 1$ for every μ if $\text{dom}_\mu = \mathit{var}(\mathcal{P})$ and $\mu(\mathcal{P}) \subseteq G$, and 0 otherwise;
2. $|\mu, \text{Proj}_L(\mathcal{P})^G| = \sum_{\mu = \mu'_{|L}} |\mu', \mathcal{P}^G|$;
3. $|\mu, \text{Dist}(\mathcal{P})^G| = \begin{cases} 1, & \text{if } |\mu, \mathcal{P}^G| > 0, \\ 0, & \text{otherwise;} \end{cases}$
4. $|\mu, \text{Filter}_\mathcal{F}(\mathcal{P})^G| = \begin{cases} |\mu, \mathcal{P}^G|, & \text{if } \mathcal{F}^\mu = \top, \\ 0, & \text{otherwise;} \end{cases}$
5. $|\mu, \text{Union}(\mathcal{P}_1, \mathcal{P}_2)^G| = |\mu, {\mathcal{P}_1}^G| + |\mu, {\mathcal{P}_2}^G|$;
6. $|\mu_1 \oplus \mu_2, \text{Join}(\mathcal{P}_1, \mathcal{P}_2)^G| = \sum_{\mu_1 \in {\mathcal{P}_1}^G, \mu_2 \in {\mathcal{P}_2}^G \text{ with } \mu_1 \sim \mu_2} |\mu_1, {\mathcal{P}_1}^G| \times |\mu_2, {\mathcal{P}_2}^G|$;
7. $|\mu, \text{Minus}(\mathcal{P}_1, \mathcal{P}_2)^G| = \begin{cases} |\mu, {\mathcal{P}_1}^G|, & \text{if } \forall \mu_2 \in {\mathcal{P}_2}^G.(\mu \nsim \mu_2 \text{ or } \text{dom}_\mu \cap \text{dom}_{\mu_2} = \emptyset), \\ 0, & \text{otherwise;} \end{cases}$
8. $|\mu, \text{Diff}_\mathcal{F}(\mathcal{P}_1, \mathcal{P}_2)^G| = \begin{cases} |\mu, {\mathcal{P}_1}^G|, & \text{if } \forall \mu_2 \in {\mathcal{P}_2}^G.(\mu \nsim \mu_2 \text{ or } \mathcal{F}^{\mu \oplus \mu_2} \neq \top), \\ 0, & \text{otherwise;} \end{cases}$
9. $|\mu, \text{Opt}_\mathcal{F}(\mathcal{P}_1, \mathcal{P}_2)^G| = |\mu, \text{Filter}_\mathcal{F}(\text{Join}(\mathcal{P}_1, \mathcal{P}_2))^G| + |\mu, \text{Diff}_\mathcal{F}(\mathcal{P}_1, \mathcal{P}_2)^G|$.

The support of a multiset M is the underlying set $\text{sup}(M) = \{\mu \mid |\mu, M| > 0\}$. The domain of a multiset M of mappings is defined as $\text{dom}_M = \bigcup_{\mu \in M} \text{dom}_\mu$. We denote by $M_{|\bar{X}}$ the multiset of mappings $\mu \in M$ restricted to $\mathcal{V} \setminus X$, i.e., $|\mu, M_{|\bar{X}}| = \sum_{\mu = \mu'_{|\bar{X}}} |\mu', M|$.

Finally, Proj and Dist operators capture the query nesting functionality, called *subqueries* [30, Sect. 3], in SPARQL, which can be arbitrarily deep and may lead to a complex layered structure. We thus adopt the *simple normal form* (SNF) of SPARQL queries from [30, Defn. 3.8] that is suitable for optimizations presented in Sect. 3.

Definition 2 *A SPARQL query is in SNF if it has the form* $\text{Dist}(\text{Proj}_X(\mathcal{P}))$ *with subquery-free pattern* $\mathcal{P}$ *or the form* $\text{Proj}_X(\mathcal{P})$*, where all subquery patterns in* $\mathcal{P}$ *are of the form* $\text{Dist}(\text{Proj}_{X'}(\mathcal{P}'))$ *with* $\mathcal{P}'$ *subquery-free.*

Any SPARQL query Q can be brought into *SNF* by following two steps of normalisation as illustrated below, where $n \geq i \geq 1$:

1. If Q is a query of the form $\text{Dist}(\text{Proj}_X(\mathcal{P}))$ s.t. $\mathcal{P}$ is subquery $\text{Dist}(\mathcal{P}')$, then Q can be simplified to $\text{Dist}(\text{Proj}_X(\mathcal{P}'))$.

2. If Q is a query of the form $\textsc{Proj}_X(\mathcal{P})$ s.t. $\mathcal{P} = \{\mathcal{P}_1 \dots \textsc{Proj}_Y(\mathcal{P}_i) \dots \mathcal{P}_n\}$, then Q can be simplified to $\textsc{Proj}_X(\mathcal{P}')$, where $\mathcal{P}' = \{\mathcal{P}_1 \dots \mathcal{P}_i{}^{W \mapsto W'} \dots \mathcal{P}_n\}$ s.t. $W = (var(P_i) \setminus Y) \cap var(Q \setminus P_i)$ and $w' \notin var(Q)$ for all $w' \in W'$.

Step 1 removes redundant `Dist` from inner graph pattern $\mathcal{P}'$ since the outermost `Dist` already guarantees a duplicate-free result. To avoid conflicts with variables outside of $\mathcal{P}_i$ in Q, step 2 renames variables in inner graph pattern $\mathcal{P}_i$ before removing the inner `Proj`.

Henceforth, we assume that all SPARQL queries are in their *SNF*.

Example 2 *Consider a SPARQL query that retrieves the name of employees and their office addresses, such as,*

$$\textsc{Proj}_{yz}(\textsc{Join}(\mathtt{hasName}(x, y), \textsc{Proj}_{xz}(\textsc{Join}(\mathtt{hasOffice}(x, y), \mathtt{hasAddress}(y, z))))),$$

which can be brought into SNF by renaming the variable y of subquery to n, thus removing clashes with any variables outside of the subquery, as follows,

$$\textsc{Proj}_{yz}(\mathtt{hasName}(x, y)\ \mathtt{hasOffice}(x, n)\ \mathtt{hasAddress}(n, z))\,.$$

SHACL Constraint. We adopt the abstract syntax for the *core constraints* of SHACL [32] introduced by Corman et al. [19]. In addition, we incorporate a non-standard SPARQL-based constraint component called `dash:uniqueValuesForClass`. Let **S**, **C** and **P** be countably infinite and mutually disjoint sets of SHACL *shape*, *class* and *property* names, respectively. Each SHACL constraint is a set of conditions, usually referred to as shape, defined as a triple $\langle s, \tau_s, \phi_s \rangle$ consisting of:

- *name s*,
- *target definition* τ_s is a SPARQL query with a single output variable, which retrieves the target entities of s from the RDF graph G that is being validated. There are two types of τ_s that we consider:
  - The first type corresponds to the ‘sh:targetClass’ of the SHACL specification. In this case, τ_s is a SPARQL query of the form:

    “SELECT ?x WHERE ?x rdf:type/subClassOf* C ”.

  - The second type corresponds to the ‘sh:targetSubjectOf’ or ‘sh:targetObjectOf’ of the SHACL specification, as shown in Example 3. In this case, τ_s is a SPARQL query of the form:

    “SELECT ?x WHERE ?x P ?y ” or “SELECT ?x WHERE ?y P ?x ”.

  In abstract syntax, we express τ_s using the following grammar:

  $$\tau_s ::= C \mid \exists P \mid \exists P^-$$

  where τ_s being $C, \exists P$ and $\exists P^-$ represent ‘sh:targetClass C’, ‘sh:targetSubjectOf P’, and ‘sh:targetObjectOf P’ in the SHACL specification, respectively. Given a target expression τ_s, we say that $n \in \mathcal{N}_G$ is a target of the shape s, written $G \models \tau_s(n)$, iff $\exists \mu \in {\tau_s}^G$ s.t. $n \in \mu(var(\tau_s))$.

- *constraint definition* ϕ_s, which can be represented as a boolean SPARQL query as described in [18,17,10], indicating whether the target node under validation violates the graph pattern defined by the ϕ_s or not. The constraint ϕ_s is an expression defined according to the following grammar:

$$\phi_s ::= \ \geq_n \alpha.\beta \ | \leq_n \alpha.\beta \ | \ \rhd_{\tau_s} \alpha \ | \ \alpha_1 = \alpha_2 \ | \ \phi_s \wedge \phi_s$$
$$\beta ::= \top \ | \ C \ | \ s' \ | \ \neg\beta$$

  where $\top$ stands for the *Boolean* true value, $\alpha, \alpha_1, \alpha_2 \in (\mathbf{P} \cup \{P^- \mid P \in \mathbf{P}\})$, $C \in \mathbf{C}$, $s' \in \mathbf{S}$, $\neg$ stands for *negation*, $n \in \mathbb{N}$, $(\geq_n \alpha.\beta)$ requires that there must be at least n α-successors verifying β, $(\rhd_{\tau_s} \alpha)$ that the value node of α-successor must be unique among target nodes defined by the τ_s, e.g., `dash:uniqueValueForClass` constraint for the class-based target, and $(\alpha_1 = \alpha_2)$ means equality of the sets of nodes reachable via the respective properties α_1 and α_2. As syntactic sugar, we write $\phi_1 \vee \phi_2$ for $\neg(\neg\phi_1 \wedge \neg\phi_2)$ and $=_n \alpha.\phi$ for $(\geq_n \alpha.\phi) \wedge (\leq_n \alpha.\phi)$. For a constraint expression ϕ_s and a node $n \in \mathcal{N}_G$, we say that n validates against ϕ_s, written $G \models \phi_s(n)$, iff n does not violate the graph pattern, aka constraints, defined by ϕ_s. For any constraints ϕ and ϕ', we say that ϕ implies ϕ', rewritten $\phi \longrightarrow \phi'$, iff for all graphs G and nodes $n \in \mathcal{N}_G$, if $G \models \phi(n)$ then $G \models \phi'(n)$.

A SHACL document is a set of SHACL shapes. An RDF graph G validates against a shape $\langle s, \tau_s, \phi_s \rangle$, if $G \models \phi_s(n)$ for all $n \in \mathcal{N}_G$ with $G \models \tau_s(n)$. An RDF graph G validates against a SHACL document $\mathcal{S}$, written $G \models \mathcal{S}$, iff G validates against all shapes in $\mathcal{S}$.

Example 3 *Assume an RDF graph on the left that validates a shape with a 'property-based target,' i.e.,* `sh:targetSubjectOf :hasID`*, on the right written in Turtle syntax:*

```
:Ida a :Student;             :TargetSubjectShape a sh:NodeShape;
  :hasID "001"^^xsd:int;      sh:targetSubjectOf  :hasID;
  :hasAddress "Oslo".         sh:property [ sh:path :hasAddress;
:Nora a :Student;                         sh:nodeKind sh:Literal;
  :hasAddress "Oslo".                     sh:minCount 1;    ];
:Ingrid a :Student;           sh:property [ sh:path  :hasID;
  :hasID "002"^^xsd:int;                   dash:uniqueValueForClass
  :hasAddress "Bergen".                             :Student ; ].
```

Observe that '`Nora`*' is not subject of a '*`hasID`*' triple, and therefore, is not a target node of the shape '*`TargetSubjectShape`*'. Since the target nodes, i.e., '*`Ida`*' and '*`Ingrid`*', of the shape have an address and unique ids, they validate against the shape.*

3 Optimizations

The goal is to optimize a SPARQL query with respect to a SHACL document, by identifying smaller or more efficient queries that have the same answers for all RDF graphs that satisfy the SHACL document.

Next we will present SPARQL query optimizations that resemble SQL query rewriting using relational constraints. They only differ in how SPARQL graph patterns interact with SHACL constraints describing these graph patterns in the RDF graph. For instance, when property path cardinality constraints imply that the first argument of an `Opt`-pattern will always find at least one solution on the second, we can substitute `Opt` with a more straightforward join operator, as shown in Lemmas 1.1, 3.1 and 6.1. In the case of a distinct projection, with an 'at least one' property path cardinality constraint, we can infer that every solution mapping from the first argument of the join expression is guaranteed to find at least one solution on the second argument. This guarantee on the graph patterns allows us to reduce the entire joint expression to the first argument, as shown in Lemmas 1.2 and 3.2. In the case of an 'exactly one' property path cardinality constraint, the described optimization of join patterns occurs even in the absence of distinct projection, as shown in Lemmas 2 and 4. If all possible mappings for the first argument of the optional pattern can find a compatible solution with the second argument, and if the variables used in filter are limited to the first argument, then it is possible to replace the entire optional pattern with the filter expression applied to the first argument, as shown in Lemma 5.2, Corollaries 2.2, 4.2 and 5.2.

Let $\mathcal{U}$ and $\mathcal{V}$ be two graph patterns and $\mathcal{S}$ a SHACL document.

Definition 3 *$\mathcal{U} \equiv_{\mathcal{S}} \mathcal{V}$ iff $\mathcal{U}^G = \mathcal{V}^G$ for all graphs G with $G \models \mathcal{S}$.*

Definition 4 *$\mathcal{U} \equiv_{\mathcal{S},y} \mathcal{V}$ iff, for all graphs G with $G \models \mathcal{S}$,*

1. *${\mathcal{U}^G}_{|\bar{y}} = {\mathcal{V}^G}_{|\bar{y}}$ and*
2. *$(\mu_{|\bar{y}} = {\mu'}_{|\bar{y}}) \longrightarrow (\mu = \mu')$ for all $\mu, \mu' \in \mathcal{U}^G$ and $\mu, \mu' \in \mathcal{V}^G$.*

Definition 5 *$\mathcal{U} \cong_{\mathcal{S},y} \mathcal{V}$ iff $sup({\mathcal{U}^G}_{|\bar{y}}) = sup({\mathcal{V}^G}_{|\bar{y}})$ for all graphs G with $G \models \mathcal{S}$.*

In Theorem 1, we will prove that an $\equiv_{\mathcal{S}}$ equivalence is applicable to all queries, while $\equiv_{\mathcal{S},y}$ is restricted to the placement of variable y within the query, and $\cong_{\mathcal{S},y}$ is exclusive to the distinct graph patterns with restrictions on the placement of variable y. In the subsequent lemmas 1 to 7, we will establish these equivalences under various constraints, and illustrate their application along with counterexamples when restrictions on the placement of variables are violated in the query.

Let $\mathcal{B}$ and $\mathcal{B}'$ be two BGPs and $\mathcal{P}$ a graph pattern. Then, we write $\mathcal{B} \blacktriangleleft \mathcal{P}$ (resp., $\mathcal{B}, \mathcal{B}' \blacktriangleleft \mathcal{P}$) if $\mathcal{P} = \mathcal{B}$ or $\mathcal{P} = \mathcal{B}\mathcal{P}_1 \ldots \mathcal{P}_n$ (resp., $\mathcal{P} = \mathcal{B}\mathcal{B}'$ or $\mathcal{P} = \mathcal{B}\mathcal{B}'\mathcal{P}_1 \ldots \mathcal{P}_n$), where $\mathcal{P}_i$ s.t. $n \geq i \geq 1$ are inner graph patterns. Let $C, C' \in \mathbf{C}$ and $P, R \in (\gamma \cup \{\gamma^- \mid \gamma \in \mathbf{P}\})$. Then, in the lemmas that follow, given an arbitrary shape $\langle s, \tau_s, \phi_s \rangle$ with target τ_s definition, let T be a triple pattern s.t.,

$$T = \begin{cases} C(x), & \text{if } \tau_s = C, \\ R(x,z), & \text{if } \tau_s = \exists R, \\ R^-(x,z), & \text{if } \tau_s = \exists R^- . \end{cases}$$

Lemma 1. *Let $\langle s, \tau_s, \phi_s \rangle \in \mathcal{S}$ with $(\geq_n P.\top) \in \phi_s$ s.t. $n \geq 1$, and $\mathcal{P}$ a graph pattern s.t. $T \blacktriangleleft \mathcal{P}$. If $y \notin var(\mathcal{P})$, then*

1. $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, P(x,y)) \equiv_{\mathcal{S}} \textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, P(x,y)))$
2. $\textsc{Join}(\mathcal{P}, P(x,y)) \cong_{\mathcal{S},y} \mathcal{P}$

Proof. Assuming $T \blacktriangleleft \mathcal{P}$ w.r.t. the target τ_s of the shape $\langle s, \tau_s, \phi_s \rangle \in \mathcal{S}$, we know that x is the only variable shared between $\mathcal{P}$ and $P(x,y)$, and y is not in $var(\mathcal{P})$. Let G be a graph s.t. $G \models \mathcal{S}$.

Clause 1: Using the definition of Opt, we can split the solutions of $\text{Opt}_{\mathcal{F}}(\mathcal{P}, P(x,y))$ into two parts: finding solutions that satisfy both $\mathcal{P}$ and $P(x,y)$, and finding solutions that satisfy only $\mathcal{P}$. For any mapping in the first part $\mathcal{P}^G$, we can find at least n mappings in the second part $P(x,y)^G$, where $n \geq 1$ is guaranteed by the constraint $(\geq_n P.\top) \in \phi_s$. Therefore, any solution that satisfies $\mathcal{P}$ and $P(x,y)$ will also be part of the solutions that satisfy only $\mathcal{P}$ w.r.t. G and vice-versa, which means that $|\mu, \text{Opt}_{\mathcal{F}}(\mathcal{P}, P(x,y))^G| = |\mu, \text{Filter}_{\mathcal{F}}(\text{Join}(\mathcal{P}, P(x,y)))^G|$. Thus, $\text{Opt}_{\mathcal{F}}(\mathcal{P}, P(x,y)) \equiv_{\mathcal{S}} \text{Filter}_{\mathcal{F}}(\text{Join}(\mathcal{P}, P(x,y)))$ by Defn. 3 if $y \notin var(\mathcal{P})$.

Clause 2: Starting from the left-hand side $\text{Join}(\mathcal{P}, P(x,y))$, we have $y \notin var(\mathcal{P})$. Using the definition of Join, the problem is to find all possible combinations of solutions that satisfy $\mathcal{P}$ and $P(x,y)$. Since constraint $(\geq_n P.\top) \in \phi_s$ together with conditions $T \blacktriangleleft \mathcal{P}$ and $y \notin var(\mathcal{P})$ guarantees that there exist least n mappings $\mu_2 \in P(x,y)^G$ for $\forall \mu_1 \in \mathcal{P}^G$ s.t. $\mu_1 \sim \mu_2$, we can deduce that any solution that satisfies $\mathcal{P}$ will also satisfy $\text{Join}(\mathcal{P}, P(x,y))$ w.r.t. G. Thus, we can claim that $\sup(\text{Join}(\mathcal{P}, P(x,y))^G_{|\bar{y}})$ is equal to $\sup(\mathcal{P}^G_{|\bar{y}})$, i.e., $\text{Join}(\mathcal{P}, P(x,y)) \cong_{\mathcal{S},y} \mathcal{P}$ by Defn 4.

Corollary 1 follows from Lemma 1.

Corollary 1 *Let $\langle s, \tau_s, \phi_s \rangle \in \mathcal{S}$ with $(\geq_n P.\top) \in \phi_s$ s.t. $n \geq 1$, and $\mathcal{P}$ a graph pattern s.t. $T \blacktriangleleft \mathcal{P}$. If $y \notin var(\mathcal{P} \cup \mathcal{F})$, then*

1. $\text{Filter}_{\mathcal{F}}(\text{Join}(\mathcal{P}, P(x,y))) \cong_{\mathcal{S},y} \text{Filter}_{\mathcal{F}}(\mathcal{P})$
2. $\text{Opt}_{\mathcal{F}}(\mathcal{P}, P(x,y)) \cong_{\mathcal{S},y} \text{Filter}_{\mathcal{F}}(\mathcal{P})$

Example 4 *Let G be a graph such that $G \models \langle \texttt{Student}, \tau_{\texttt{Student}}, \phi_{\texttt{Student}} \rangle$ with $(\geq_n$ $\texttt{:hasAddress}.\top) \in \phi_{\texttt{Student}}$ s.t. $n \geq 1$. Then, let Q be a query that asks for all students who do not have a postal address, where $\top$ denotes the tautological filter (true),*

$$\text{Proj}_x(\text{Filter}_{\neg bound(y)}(\text{Opt}_\top(\texttt{Student}(x), \texttt{hasAddress}(x,y))))$$

over G. As per the constraint $\phi_{\texttt{Student}}$, all students in G have a postal address. Hence, the optimal solution would be to replace the entire query with an empty query. However, in our setting, the Opt*-pattern of Q can be reduced to an equivalent* Join*-pattern by following the clause 1 of Lemma 1 since y does not appear in the first part of the* Opt*-pattern,*

$$\text{Proj}_x(\text{Filter}_{\neg bound(y)}(\text{Join}(\texttt{Student}(x), \texttt{hasAddress}(x,y)))).$$

In a scenario where y also appears in the first part of the Opt*-pattern, such as*

$$\text{Proj}_x(\text{Filter}_{\neg bound(y)}(\text{Opt}_\top(\texttt{Student}(x)\ \texttt{enrolledIn}(x,y), \texttt{hasAddress}(x,y)))).$$

A counter-example $G = \{\texttt{Student}(a), \texttt{enrolledIn}(a,b), \texttt{hasAddress}(a,d)\}$ s.t. $G \models \langle \texttt{Student}, \tau_{\texttt{Student}}, \phi_{\texttt{Student}} \rangle$ can be constructed, where a solution can be found that yields "$\texttt{Student(a)}$" for the Opt *pattern but none when* Opt *is reduced to* Join*-pattern.*

Similarly, assume a query that asks for students who have postal addresses, such as

$$\textsc{Dist}(\textsc{Proj}_{x\,y}(\textsc{Join}(\mathtt{Student}(x)\ \mathtt{hasName}(x,y), \mathtt{hasAddress}(x,z))))$$

over the graph G. Then, clause 2 of Lemma 1 allows us to safely remove the pattern $\mathtt{hasAddress}(x,z)$ *from the '*Dist*' query, given that variable z only appears in this pattern and all students in the graph G satisfying* $\langle \mathtt{Student}, \tau_{\mathtt{Student}}, \phi_{\mathtt{Student}} \rangle$ *have at least n postal addresses.,i.e.,*

$$\textsc{Dist}(\textsc{Proj}_{x\,y}(\mathtt{Student}(x)\ \mathtt{hasName}(x,y)))\,.$$

However, a counter-example can be found for the $\cong_{\mathcal{S},y}$ *equivalence when variable z appears outside of the pattern* $\mathtt{hasAddress}(x,z)$ *in the* Dist *query above, such as*

$$\textsc{Dist}(\textsc{Proj}_{x\,y}(\textsc{Filter}_{\neg bound(z)}(\textsc{Join}(\mathtt{Student}(x)\ \mathtt{hasName}(x,y), \mathtt{hasAddress}(x,z))))).$$

Finally, assume a query scenario over the G as per required by the 'if-condition' in Corollary 1,

$$\textsc{Dist}(\textsc{Proj}_{x\,z}(\textsc{Opt}_{z=\text{'}A\text{'}}(\mathtt{Student}(x)\ \mathtt{hasGrade}(x,z), \mathtt{hasAddress}(x,y)))).$$

Then, the equivalent "$\textsc{Dist}(\textsc{Proj}_{x\,z}(\textsc{Filter}_{z=\text{'}A\text{'}}(\mathtt{Student}(x)\ \mathtt{hasGrade}(x,z))))$*" query can be deduced by using Corollary 1.2, which reduces the entire '*Opt*'-pattern to the first part of the pattern with the filter applied (i.e.,* $\cong_{\mathcal{S},y}$ *equivalence).*

Lemma 2 follows from the same reasoning as Lemma 1.

Lemma 2. *Let* $\langle s, \tau_s, \phi_s \rangle \in \mathcal{S}$ *with* $(=_1\ P.\top) \in \phi_s$ *and* $\mathcal{P}$ *a graph pattern s.t.* $T \blacktriangleleft \mathcal{P}$*. If* $y \notin var(\mathcal{P})$*, then* $\textsc{Join}(\mathcal{P}, P(x,y)) \equiv_{\mathcal{S},y} \mathcal{P}$*.*

Corollary 2 follows from Lemma 1 and Lemma 2.

Corollary 2 *Let* $\langle s, \tau_s, \phi_s \rangle \in \mathcal{S}$ *with* $(=_1\ P.\top) \in \phi_s$ *s.t.* $n \geq 1$*, and* $\mathcal{P}$ *a graph pattern s.t.* $T \blacktriangleleft \mathcal{P}$*. If* $y \notin var(\mathcal{P} \cup \mathcal{F})$*, then*

1. $\textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, P(x,y))) \equiv_{\mathcal{S},y} \textsc{Filter}_{\mathcal{F}}(\mathcal{P})$
2. $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, P(x,y)) \equiv_{\mathcal{S},y} \textsc{Filter}_{\mathcal{F}}(\mathcal{P})$

Example 5 *consider a SPARQL query that asks for the names of those who have postal addresses, such as*

$$\textsc{Proj}_{y}(\textsc{Join}(\mathtt{hasName}(x,y), \mathtt{hasAddress}(x,z)))\,,$$

over a graph G s.t. $G \models \langle \exists\mathtt{hasName}, \tau_{\exists\mathtt{hasName}}, \phi_{\exists\mathtt{hasName}} \rangle$ *and* $(=_1\ \mathtt{:hasAddress}.\top) \in \phi_{\exists\mathtt{hasName}}$*. Since the constraint* $\phi_{\exists\mathtt{hasName}}$ *ensures that individuals with names have exactly one postal address, we can deduce the following equivalent query based on Lemma 2:*

$$\textsc{Proj}_{y}(\mathtt{hasName}(x,y))\,.$$

In Lemmas 3 to 5,

- let β be either C' or s' s.t. $s' \in \mathcal{S}$, and
- let $\mathcal{P}'$ be a graph pattern s.t. $\mathcal{P}'$ is either $P(x,y)$ or $P(x,y)C'(y)$ if $\beta = s'$, and $P(x,y)C'(y)$ if $\beta = C'$.

Lemma 3. *Let $\langle s, \tau_s, \phi_s \rangle \in S$ with $(\geq_n P.\beta) \in \phi_s$ s.t. $n \geq 1$, and $\mathcal{P}$ a graph pattern s.t. $T \blacktriangleleft \mathcal{P}$. If $y \notin var(\mathcal{P})$, then*

1. $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, \mathcal{P}') \equiv_{\mathcal{S}} \textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, \mathcal{P}'))$
2. $\textsc{Join}(\mathcal{P}, \mathcal{P}') \cong_{\mathcal{S},y} \mathcal{P}$

In contrast to Lemma 1, Lemma 3 also handles type constraints on the value node of a property predicate P. Corollary 3 follows from Lemma 3.

Corollary 3 *Let $\langle s, \tau_s, \phi_s \rangle \in S$ with $(\geq_n P.\beta) \in \phi_s$ s.t. $n \geq 1$, and $\mathcal{P}$ a graph pattern s.t. $T \blacktriangleleft \mathcal{P}$. If $y \notin var(\mathcal{P} \cup \mathcal{F})$, then*

1. $\textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, \mathcal{P}')) \cong_{\mathcal{S},y} \textsc{Filter}_{\mathcal{F}}(\mathcal{P})$
2. $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, \mathcal{P}') \cong_{\mathcal{S},y} \textsc{Filter}_{\mathcal{F}}(\mathcal{P})$

Example 6 *Suppose we have a SPARQL query that asks the names of individuals who have valid addresses, such as*

$$\textsc{Dist}(\textsc{Proj}_y(\textsc{Join}(\texttt{hasName}(x, y), \texttt{hasAddress}(x, z)\ \texttt{Address}(z)))),$$

over a graph G s.t. $G \models \langle \exists\texttt{hasName}, \tau_{\exists\texttt{hasName}}, \phi_{\exists\texttt{hasName}} \rangle$ and $(\geq_1 \texttt{:hasAddress}.\texttt{Address}) \in \phi_{\texttt{People}}$. Then, we can use Lemma 3, specifically clause 2, to simplify the query to:

$$\textsc{Dist}(\textsc{Proj}_y(\texttt{hasName}(x, y))).$$

Lemma 4. *Let $\langle s, \tau_s, \phi_s \rangle \in S$ with $(=_1 P.\beta) \in \phi_s$, and $\mathcal{P}$ a graph pattern s.t. $T \blacktriangleleft \mathcal{P}$. If $y \notin var(\mathcal{P})$, then $\textsc{Join}(\mathcal{P}, \mathcal{P}') \equiv_{\mathcal{S},y} \mathcal{P}$.*

Lemma 4 follows from the same reasoning as Lemma 3, and Corollary 4 follows from Lemma 3 and Lemma 4.

Corollary 4 *Let $\langle s, \tau_s, \phi_s \rangle \in S$ with $(=_1 P.\beta) \in \phi_s$ s.t. $n \geq 1$, and $\mathcal{P}$ a graph pattern s.t. $T \blacktriangleleft \mathcal{P}$. If $y \notin var(\mathcal{P} \cup \mathcal{F})$, then*

1. $\textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, \mathcal{P}')) \equiv_{\mathcal{S},y} \textsc{Filter}_{\mathcal{F}}(\mathcal{P})$
2. $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, \mathcal{P}') \equiv_{\mathcal{S},y} \textsc{Filter}_{\mathcal{F}}(\mathcal{P})$

Lemma 5. *Let $\langle s, \tau_s, \phi_s \rangle \in S$ with $(\geq_n P.\beta) \in \phi_s$ s.t. $n \geq 1$ or $(\leq_0 P.\neg\beta) \in \phi_s$, and $\mathcal{P}$ a graph pattern s.t. $T, P(x, y) \blacktriangleleft \mathcal{P}$. Then,*

1. $\textsc{Join}(\mathcal{P}, C'(y)) \equiv_{\mathcal{S}} \mathcal{P}$
2. $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, C'(y)) \equiv_{\mathcal{S}} \textsc{Filter}_{\mathcal{F}}(\mathcal{P})$

Proof. Assuming $T, P(x, y) \blacktriangleleft \mathcal{P}$ w.r.t. the target τ_s of the shape $\langle s, \tau_s, \phi_s \rangle \in \mathcal{S}$, we have $var(\mathcal{P}) \cap var(C'(y)) = \{y\}$. Let G be a graph s.t. $G \models \mathcal{S}$. Then,

Clause 1: Starting from the left-hand side pattern $\textsc{Join}(\mathcal{P}, C'(y))$, the constraint $(\geq_n P.\beta) \in \phi_s$ (resp., $(\leq_0 P.\neg\beta) \in \phi_s$) s.t. $n \geq 1$ together with the condition $P(x, y) \blacktriangleleft \mathcal{P}$ guarantees that there exists exactly one mapping $\mu_2 \in C'(y)^G$ for $\forall \mu_1 \in \mathcal{P}^G$ s.t. $\mu_1 \sim \mu_2$. Thus, we can deduce $|\mu, \textsc{Join}(\mathcal{P}, P(x, y))^G| = |\mu, \mathcal{P}^G|$, which implies $\textsc{Join}(\mathcal{P}, P(x, y)) \equiv_{\mathcal{S}} \mathcal{P}$ by Defn 3.

Clause 2: For the pattern $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, C'(y))$, constraint $(\geq_n\ P.\beta) \in \phi_s$ (resp., $(\leq_0 P.\neg\beta) \in \phi_s$) together with the condition $P(x, y) \blacktriangleleft \mathcal{P}$ guarantees that for any mapping in the first part $\mathcal{P}^G$, we can find exactly one mapping in the second part $C'(y)^G$. Thus, any solution that satisfies $\mathcal{P}$ and $C'(y)$ will also be part of the solutions that satisfy only $\mathcal{P}$ w.r.t. G and vice-versa, which means that $|\mu, \textsc{Opt}_{\mathcal{F}}(\mathcal{P}, C'(y))^G| = |\mu, \textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, C'(y)))^G|$, i.e., $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, C'(y)) \equiv_{\mathcal{S}} \textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, C'(y)))$ by Defn. 3.

Further, the restriction of solutions to $\mathcal{F}^\mu = \top$ is identical for both $\textsc{Join}(\mathcal{P}, C'(y))^G$ and $\mathcal{P}^G$ since $var(\mathcal{F}) \subseteq var(\mathcal{P})$ can be deduced from the fact $var(C'(y)) \subseteq var(\mathcal{P})$. Then, using clause 1, i.e., $\textsc{Join}(\mathcal{P}, C'(y)) \equiv_{\mathcal{S}} \mathcal{P}$, we can infer that $\textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, C'(y))) \equiv_{\mathcal{S}} \textsc{Filter}_{\mathcal{F}}(\mathcal{P})$. Thus,

$$\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, C'(y)) \equiv_{\mathcal{S}} \textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, C'(y))) \equiv_{\mathcal{S}} \textsc{Filter}_{\mathcal{F}}(\mathcal{P}) .$$

Example 7 *Let G be a graph that satisfies the shapes* $\langle \exists\texttt{hasName}, \tau_{\exists\texttt{hasName}}, \phi_{\exists\texttt{hasName}}\rangle$ *and* $\langle \texttt{Program}, \tau_{\texttt{Program}}, \phi_{\texttt{Program}}\rangle$*, where* $(\leq_0\ \texttt{:enrolledIn}.\neg, \texttt{Program}) \in \phi_{\exists\texttt{hasName}}$*. Consider the query,*

$$\textsc{Proj}_{xz}(\textsc{Opt}_{\top}(\texttt{hasName}(x, y)\ \texttt{enrolledIn}(x, z),\ \texttt{Program}(z)))$$

over the G. According to the constraint $\phi_{\exists\texttt{hasName}}$*, all enrollments are only in the '*`Program`*'. Hence, using clause 1 and 2 of Lemma 5, we can obtain the following equivalent query:*

$$\textsc{Proj}_{xz}(\texttt{hasName}(x, y)\ \texttt{enrolledIn}(x, z)) .$$

Given a shape $\langle s, \tau_s, \phi_s\rangle \in \mathcal{S}$, let $\mathcal{K}$ be a graph pattern s.t., $\mathcal{K} = \begin{cases} C(x)R(x,z), & \text{if } \tau_s = C, \\ R(x,z), & \text{if } \tau_s = \exists R, \\ R^-(x,z), & \text{if } \tau_s = \exists R^- . \end{cases}$

Lemma 6. *Let* $\langle s, \tau_s, \phi_s\rangle \in S$ *with* $(R = P) \in \phi_s$*, and* $\mathcal{P}$ *a graph pattern s.t.* $\mathcal{K} \blacktriangleleft \mathcal{P}$*. If* $y \notin var(\mathcal{P})$*, then*

1. $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, P(x, y)) \equiv_{\mathcal{S}} \textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, P(x, y)))$
2. $\textsc{Join}(\mathcal{P}, P(x, y)) \cong_{\mathcal{S},y} \mathcal{P}$

Proof. Assuming $\mathcal{K} \blacktriangleleft \mathcal{P}$ w.r.t. the target τ_s of the shape $\langle s, \tau_s, \phi_s\rangle \in \mathcal{S}$, we have $var(\mathcal{P}) \cap var(P(x, y)) = \{x\}$ since $R(x, z) \blacktriangleleft \mathcal{K}$. Let G be a graph s.t. $G \models \mathcal{S}$.

Clause 1: For $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, P(x, y))$ s.t. $y \notin var(\mathcal{P})$, the constraint $(\mathtt{R} = \mathtt{P}) \in \phi_s$ together with the condition $R(x, z) \blacktriangleleft \mathcal{P}$ guarantees that for any mapping in the first part $\mathcal{P}^G$, we can find exactly one mapping in the second part $P(x, y)^G$. Thus, any solution that satisfies $\mathcal{P}$ and $P(x, y)$ will also be part of the solutions that satisfy only $\mathcal{P}$ w.r.t. G and vice-versa, which means that $|\mu, \textsc{Opt}_{\mathcal{F}}(\mathcal{P}, P(x, y))^G| = |\mu, \textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, P(x, y)))^G|$ if $y \notin var(\mathcal{P})$, i.e.,

$$\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, P(x, y)) \equiv_{\mathcal{S}} \textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, P(x, y))) \text{ by Defn. 3.}$$

Clause 2: Consider the left-hand side $\textsc{Join}(\mathcal{P}, P(x, y))$ s.t. $y \notin var(\mathcal{P})$. Then, the constraint $(\mathtt{R} = \mathtt{P}) \in \phi_s$ together with the condition $R(x, z) \blacktriangleleft \mathcal{P}$ guarantees that there exists exactly one mapping $\mu_2 \in P(x, y)^G$ for $\forall\mu_1 \in \mathcal{P}^G$ s.t. $\mu_1 \sim \mu_2$. From this, we can deduce that $\sup(\textsc{Join}(\mathcal{P}, P(x, y))^G) = \sup(\mathcal{P}^G)$, which in turn implies $\textsc{Join}(\mathcal{P}, P(x, y)) \cong_{\mathcal{S},y} \mathcal{P}$ by Defn. 5, if $y \notin var(\mathcal{P})$.

Corollary 5 follows from Lemma 6.

Corollary 5 *Let $\langle s, \tau_s, \phi_s \rangle \in S$ with $(R = P) \in \phi_s$, and $\mathcal{P}$ a graph pattern s.t. $\mathcal{K} \blacktriangleleft \mathcal{P}$. If $y \notin var(\mathcal{P} \cup \mathcal{F})$, then*

1. $\textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, P(x, y))) \cong_{\mathcal{S},y} \textsc{Filter}_{\mathcal{F}}(\mathcal{P})$
2. $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, P(x, y)) \cong_{\mathcal{S},y} \textsc{Filter}_{\mathcal{F}}(\mathcal{P})$

Example 8 *Consider a SPARQL query,*

$$\textsc{Dist}(\textsc{Proj}_{x\,y}(\textsc{Join}(\texttt{Employee}(x)\ \texttt{insuredBy}(x, y), \texttt{employedBy}(x, z))))$$

over G s.t. $G \models \langle \texttt{Employee}, \tau_{\texttt{Employee}}, \phi_{\texttt{Employee}} \rangle$ with $(\texttt{insuredBy} = \texttt{employedBy}) \in \phi_{\texttt{Employee}}$. Then, using clause 2 of Lemma 6, we can obtain the following equivalent query:

$$\textsc{Dist}(\textsc{Proj}_{x\,y}(\texttt{Employee}(x)\ \texttt{insuredBy}(x, y)))\,.$$

Similarly, assume an optional query scenario where variable z only occurs within the pattern '$\texttt{employedBy}(x, z)$,' such as

$$\textsc{Dist}(\textsc{Proj}_{x\,y}(\textsc{Opt}_{\top}(\texttt{Employee}(x)\ \texttt{insuredBy}(x, y), \texttt{employedBy}(x, z))))\,.$$

*Using Corollary 5.2, which applies $\cong_{\mathcal{S},y}$ equivalence to reduce the entire '*Opt*'-pattern to the first part of the Opt-pattern with the filter applied, we can deduce the following equivalent query:*

$$\textsc{Dist}(\textsc{Proj}_{x\,y}(\textsc{Filter}_{\top}(\texttt{Employee}(x)\ \texttt{insuredBy}(x, y))))\,.$$

In lemma 7, let $\phi' = \begin{cases} \top, & \text{if } \mathcal{P} = T, \\ \bigwedge_{i=1}^{n} (\leq_1 P_i.\top), & \text{if } \mathcal{P} = (T\ P_1(x, z_1) \ldots P_i(x, z_i) \ldots P_n(x, z_n)). \end{cases}$

Lemma 7. *Let $\langle s, \tau_s, \phi_s \rangle \in S$ with $\{(\rhd_{\tau_s} P), \phi'\} \subseteq \phi_s$ and $\mathcal{P}$ a graph pattern s.t. $\mathcal{P} = T$ or $\mathcal{P} = (T\ P_1(x, z_1) \ldots P_i(x, z_i) \ldots P_n(x, z_n))$ for $n \geq i \geq 1$. Then,*

$$\textsc{Dist}(\textsc{Proj}_X(\textsc{Join}(\mathcal{P}, P(x, y)))) \equiv_{\mathcal{S}} \textsc{Proj}_X(\textsc{Join}(\mathcal{P}, P(x, y))).$$

Proof. For the pattern $\textsc{Proj}_X(\textsc{Join}(\mathcal{P}, P(x, y)))$, we can observe that $var(\mathcal{P}) \cap var(P(x, y)) = \{x\}$ because: (a) $\mathcal{P} = T$ or $T \blacktriangleleft \mathcal{P}$, and (b) T is either $C(x)$ or $R(x, z)$ or $R^-(x, z)$ based on the target τ_s of a shape $\langle s, \tau_s, \phi_s \rangle \in \mathcal{S}$. Using the definition of Join and Proj, the solution of $\textsc{Proj}_X(\textsc{Join}(\mathcal{P}, P(x, y)))$ is all possible combinations of solutions that satisfy $\mathcal{P}$ and $P(x, y)$, and restricted to variable set X. Let G be a graph s.t. $G \models \mathcal{S}$. Then, constraint $(\rhd_{\tau_s} P) \in \phi_s$ ensures that all P-successors binding to y are unique (i.e., distinct) among the target nodes τ_s over G. Therefore, for each mapping in $\mathcal{P}^G$, we can only find at most one corresponding mapping in $P(x, y)^G$ when $\mathcal{P} = T$. This implies $|\mu, \textsc{Proj}_X(\textsc{Join}(\mathcal{P}, P(x, y)))^G| = 1$ for $\forall \mu \in \text{sup}(\textsc{Proj}_X(\textsc{Join}(\mathcal{P}, P(x, y)))^G)$. Hence, $\textsc{Dist}(\textsc{Proj}_X(\textsc{Join}(\mathcal{P}, P(x, y)))) \equiv_{\mathcal{S}} \textsc{Proj}_X(\textsc{Join}(\mathcal{P}, P(x, y)))$.

In the case of $\mathcal{P} = \{T\ P_1(x, z_1) \ldots P_i(x, z_i) \ldots P_n(x, z_n)\}$ s.t. $n \geq 1$, the constraint component $\bigwedge_{i=1}^{n} (\leq_1 P_i.\top) \in \phi_s$ guarantees the absence of duplicate mappings $\mu \in \textsc{Proj}_X(\textsc{Join}(\mathcal{P}, P(x, y)))^G$ that could have originated from the join operations with the (inner) patterns $P_1(x, z_1) \ldots P_i(x, z_i) \ldots P_n(x, z_n)$. Thus, equivalence $\equiv_{\mathcal{S}}$ holds.

Example 9 *Consider the SPARQL query,*

$$\textsc{Dist}(\textsc{Proj}_{yz}(\texttt{Employee}(x)\ \texttt{hasName}(x,y)\ \texttt{hasID}(x,z)))$$

over a graph $\mathcal{G} \models \langle \texttt{Employee}, \tau_{\texttt{Employee}}, \phi_{\texttt{Employee}} \rangle$ *such that* $\{(\rhd_{\texttt{Employee}}\ \texttt{hasID}), (\leq_1\ \texttt{hasName}.\top)\} \subseteq \phi_{\texttt{Employee}}$ *(resp.,* $\mathcal{G} \models \langle \exists\texttt{hasName}, \tau_{\exists\texttt{hasName}}, \phi_{\exists\texttt{hasName}} \rangle$ *s.t.* $\{(\rhd_{\texttt{Employee}}\ \texttt{hasID}), (\leq_1\ \texttt{hasName}.\top)\} \subseteq \phi_{\exists\texttt{hasName}}$*). Then, '*Dist*' modifier of the query can be removed by using Lemma 7,*

$$\textsc{Proj}_{yz}(\texttt{Employee}(x)\ \texttt{hasName}(x,y)\ \texttt{hasID}(x,z))\,.$$

Definition 6 *Let Q be a SPARQL query, and let $\mathcal{P}$ and $\mathcal{U}$ be two graph patterns. Then, we write $U \trianglelefteq\!\!\!\!\sim Q$ if* $\textsc{Dist}(\textsc{Proj}_X(\mathcal{P})) \trianglelefteq Q$ *and* $\mathcal{U} \trianglelefteq \mathcal{P}$*.*

Theorem 1 specifies query rewriting, outlining the necessary conditions for applying and propagating the established equivalences from Lemmas 1 to 7 to a larger query.

Theorem 1. *Let Q be a SPARQL query and $\mathcal{S}$ a SHACL document. Let $\mathcal{U}$ and $\mathcal{V}$ be two graph patterns. Then,*

1. $Q \equiv_{\mathcal{S}} Q^{\mathcal{U}\mapsto\mathcal{V}}$ *if* $\mathcal{U} \equiv_{\mathcal{S}} \mathcal{V}$
2. $\textsc{Proj}_X(Q) \equiv_{\mathcal{S}} \textsc{Proj}_X(Q)^{\mathcal{U}\mapsto\mathcal{V}}$ *if* $\mathcal{U} \trianglelefteq Q$, $\mathcal{U} \equiv_{\mathcal{S},y} \mathcal{V}$ *and* $y \notin var(\textsc{Proj}_X(Q) \setminus \mathcal{U})$
3. $Q \equiv_{\mathcal{S}} Q^{\mathcal{U}\mapsto\mathcal{V}}$ *if* $\mathcal{U} \trianglelefteq\!\!\!\!\sim Q$, $\mathcal{U} \cong_{\mathcal{S},y} \mathcal{V}$ *and* $y \notin var(Q \setminus \mathcal{U})$

Example 10 *Consider a query that asks for all distinct combinations of student ids and their corresponding advisors (if available), for those who are currently enrolled in a university program, such as*

$$\textsc{Dist}(\textsc{Proj}_{zk}(\textsc{Opt}_\top(\texttt{Student}(x)\ \texttt{enrolledIn}(x,y)\ \texttt{Program}(y)\ \texttt{hasID}(x,z),\ \texttt{hasAdvisor}(x,k))))$$

over a graph G s.t. $G \models \langle \texttt{Student}, \tau_{\texttt{Student}}, \phi_{\texttt{Student}} \rangle$ *with* $\{(\geq_1\ \texttt{enrolledIn}.\texttt{Program}), (\rhd_{\texttt{Student}}\ \texttt{hasID}), (=_1\ \texttt{hasAdvisor}.\top)\} \subseteq \phi_{\texttt{Student}}$*. Then, using clause 1 of Lemma 1 (i.e.,* $\equiv_{\mathcal{S}}$*), we can reduce '*Opt*' to '*Join*' pattern as follows,*

$$\textsc{Dist}(\textsc{Proj}_{zk}(\textsc{Join}(\texttt{Student}(x)\ \texttt{enrolledIn}(x,y)\ \texttt{Program}(y)\ \texttt{hasID}(x,z),\ \texttt{hasAdvisor}(x,k))))\,.$$

Similarly, using clause 2 of Lemma 3 (i.e., $\cong_{\mathcal{S},y}$*), we can remove the join clause '*$\texttt{enrolledIn}(x,y)\ \texttt{Program}(y)$*' from the query,*

$$\textsc{Dist}(\textsc{Proj}_{zk}(\textsc{Join}(\texttt{Student}(x)\ \texttt{hasID}(x,z), \texttt{hasAdvisor}(x,k))))\,.$$

*Finally, '*Dist*' modifier can be removed by using Lemma 7 (i.e.,* $\equiv_{\mathcal{S}}$*),*

$$\textsc{Proj}_{zk}(\textsc{Join}(\texttt{Student}(x)\ \texttt{hasID}(x,z), \texttt{hasAdvisor}(x,k)))\,.$$

Theorems 2 and 3 state the *confluence* [8, Chapter 6] property, which guarantees that the query rewriting, defined by employing equivalences from Lemmas 1-7, is deterministic and results in the same optimized query, regardless of the order in which the rewriting rules (i.e., replacing left-hand-side of equivalences with the right-hand-side) are applied. In this context, the query rewriting defined by Lemmas 1-6 (resp., 1-7) freely employs equivalence $\equiv_{\mathcal{S}}$ for rewriting graph patterns anywhere within a query, permits the use of equivalence $\equiv_{\mathcal{S},y}$ when y is confined within the joint (or optional) triples pattern to be removed (or transformed into a join), and exclusively employs equivalence $\cong_{\mathcal{S},y}$ for rewriting distinct patterns, as described in Theorem 1.

Theorem 2. *Query rewriting defined by Lemmas 1 to 6 is a confluent reduction.*

Proof. The main point of the proof is to establish that all overlapping rewriting rules (i.e., application of equivalences) defined by Lemmas 1 to 6 always yield the same uniquely simplified pattern when applied to a graph pattern. Taking $\mathcal{P}' = P(x, y)$ into account, observe that satisfying the constraint $\phi = (\geq_n P.\beta)$ with $\beta = s'$ in Lemma 3.1 also satisfies $(\geq_n P.\top)$ in Lemma 1.1, and applying either lemma to an optional pattern of the form $\textsc{Opt}_{\mathcal{F}}(\mathcal{P}, P(x, y))$ where $y \notin var(\mathcal{P})$ results in the same pattern, namely $\textsc{Filter}_{\mathcal{F}}(\textsc{Join}(\mathcal{P}, P(x, y)))$. Similarly, fulfilling $(=_1 P.\beta)$ s.t. $\beta = s'$ in Lemma 4 satisfies $(=_1 P.\top)$ in Lemma 2, and applying either lemma leads to the same optimized join pattern. This property also applies to the implied constraint case, e.g., $(=_1 P.\top) \longrightarrow (\geq_1 P.\top)$. Likewise, satisfying ϕ in Lemma 3.2 satisfies the condition $(\geq_n P.\top)$ in Lemma 1.2, and applying either lemma produces the same optimized joint pattern.

Due to the implication $(=_1 P.\top) \longrightarrow (\geq_1 P.\top)$, when the condition of Lemma 2 is fulfilled, the rewriting of Lemma 1.2 also applies. As Lemma 2 can be applied to all patterns (Clause 1 of Theorem 2), applying both rewritings to a distinct query pattern leads to the same optimized pattern. A similar situation arises for Lemmas 2.2 and 4 due to the implication $(=_1 P.\beta) \longrightarrow (\geq_1 P.\beta)$, and they also lead to the same optimized pattern.

Theorem 3. *Query rewriting defined by Lemmas 1 to 7 is a confluent reduction iff* $\phi' = \begin{cases} \top, & \text{if } \mathcal{P} = T, \\ \bigwedge_{i=1}^{n} (=_1 P_i.\top), & \text{if } \mathcal{P} = (T\, P_1(x, z_1) \ldots P_i(x, z_i) \ldots P_n(x, z_n)) \end{cases}$ *in Lemma 7.*

The rewriting employing equivalence supported by Lemma 7 transforms a distinct pattern into a distinct-free pattern. However, certain rewritings, such as as Lemmas 1.2, 3.2 and 6.2, are exclusive to distinct patterns. Hence, the introduction of constraint $\bigwedge_{i=1}^{n} (=_1 P_i.\top)$ in Theorem 3 becomes necessary to ensure the application of complementary rewritings, particularly based on equivalences supported by Lemmas 2 and 4, when the aforementioned rewritings cannot be directly applied after Lemma 7.

Example 11 *Consider a SPARQL query,*

$$\textsc{Dist}(\textsc{Proj}_{xy}(\texttt{employeeID}(x, y)\ \texttt{hiredBy}(x, k)\ \texttt{insuredBy}(x, z)))$$

over a graph G s.t. $G \models \langle \exists\texttt{employeeID}, \tau_{\exists\texttt{employeeID}}, \phi_{\exists\texttt{employeeID}} \rangle$ with $\{(\rhd_{\exists\texttt{employeeID}}\ \texttt{employeeID}), (=_1 \texttt{insuredBy}.\top), (\texttt{hiredBy} = \texttt{insuredBy})\} \subseteq \phi_{\exists\texttt{employedID}}$. Subsequently, the constraint $(=_1 \texttt{hiredBy}.\top)$ can be inferred from the $(=_1 \texttt{insuredBy}.\top)$ and $(\texttt{hiredBy} = \texttt{insuredBy})$. Based on Theorem 1, the query is subject to optimization using the lemmas 1.2 (i.e., $\cong_{\mathcal{S},y}$), 2 (i.e., $\equiv_{\mathcal{S},y}$), 6.2 (i.e., $\cong_{\mathcal{S},y}$) and 7 (i.e., $\equiv_{\mathcal{S}}$). Then, there are seven possible rewriting combinations that can be applied using the lemmas mentioned above: (lemmas(1.2,1.2,7), lemmas(1.2,2,7), lemmas(2,1.2,7), lemmas(2,2,7), lemmas(6.2,1.2,7), lemmas(6.2,2,7), and lemmas(7,2,2)). All these combinations result in a uniquely optimized query, which is

$$\textsc{Proj}_{xy}(\texttt{employeeID}(x, y)).$$

Note that lemmas 1.2 and 6.2 cannot be applied after lemma 7. This information is further useful in determining the cheapest order in complex query optimization, which is a topic related to our future goals.

4 Discussion

We have presented a set of query rewriting rules that exploit the relation between the SPARQL queries containing graph patterns and SHACL constraints describing these graph patterns in the RDF. By treating each lemma as a set of rules under distinct constraints, we emphasized the significance of each constraint-enabled rule and made it easier to track and analyze their order and individual contributions to the overall robust query rewriting process, see Exam. 10. They also offer a clearer understanding of the different orders and strategies that can be employed while rewriting complex queries, see Exam. 11.

Constraints, known for restricting data to useful relations [23], can be used to rewrite queries into more efficient equivalents [50]. Leveraging constraints for optimizing queries has become a well-established practice in knowledge graphs [13,20,56], deductive databases [14], relational databases [25,3] and other fields [48]. Regarding SHACL, Rabbani et al. [47] were the first to propose shapes as global statistics for estimating cardinality and optimizing SPARQL query plans for the RDF graph under query. Further, Abbas et al. [1] studied the containment of restricted classes of SPARQL queries under set semantics, while considering ShEx constraints [45]. They also leveraged information obtained from ShEx schemas to optimize the execution of SPARQL queries [2]. In their work [2] (resp., [1]), a notion of well-formed ShEx schemas (resp., query patterns), especially suitable for query optimization (resp., containment), was first introduced, then subsequently used to reorganise execution orders (resp., to determine subsumption) of triple patterns in a SPARQL query. In contrast to previous studies [47,1,2], we have prioritized optimizing a much larger class of queries, and subsequently, introduced an explicit set of core rewriting rules for optimizing SPARQL queries under bag semantics when SHACL constraints are satisfied, and proved their correctness. We have further shown that our proposed query rewriting rules are confluent and always lead to a unique simplified form. We plan to investigate the potential benefits of integrating these rewriting rules into a query processing engine to enhance query execution in the near future. Additionally, we intend to extend our results to cover the broader expressivity and more general cases of SPARQL queries.

5 Conclusion

In this paper, we have proposed a set of optimizations for SPARQL query constructed from core operators of SPARQL 1.1 under the SHACL constraints. We believe that the proposed optimizations could play a crucial role in simplifying and answering SPARQL queries over large-scale RDF triples when SHACL descriptions of the underlying RDF dataset are available. In future, we aim to investigate an optimal normal form of SPARQL 1.1 [24] queries studied in [30,12] (which includes nesting, assignment, construct [35,44] and aggregation as in online analytical processing cube and window-based queries) with respect to (full) SHACL constraints [32] formalized in [19,42,41] that should allow us to introduce a confluent set of query rewriting [50, Sect. 5] rules.

Acknowledgements This work is supported by the Norwegian Research Council via the SIRIUS SFI (237898). We thank Egor Kostylev for many fruitful discussions.

References

1. Abdullah Abbas, Pierre Genevès, Cécile Roisin, and Nabil Layaïda. SPARQL query containment with ShEx constraints. In *Advances in Databases and Information Systems: 21st European Conference, ADBIS 2017, Nicosia, Cyprus, September 24-27, 2017, Proceedings 21*, pages 343–356. Springer, 2017.
2. Abdullah Abbas, Pierre Genevès, Cécile Roisin, and Nabil Layaïda. Selectivity estimation for SPARQL triple patterns with shape expressions. In *International Conference on Web Engineering*, pages 195–209. Springer, 2018.
3. Serge Abiteboul, Richard Hull, and Victor Vianu. *Foundations of databases*, volume 8. Addison-Wesley Reading, 1995.
4. Shqiponja Ahmetaj, Robert David, Axel Polleres, and Mantas Šimkus. Repairing SHACL constraint violations using answer set programming. In *International Semantic Web Conference*, pages 375–391. Springer, 2022.
5. Renzo Angles and Claudio Gutierrez. The multiset semantics of SPARQL patterns. In *International semantic web conference*, pages 20–36. Springer, 2016.
6. Mario Arias, Javier D Fernández, Miguel A Martínez-Prieto, and Pablo de la Fuente. An empirical study of real-world SPARQL queries. *arXiv preprint `https://arxiv.org/pdf/1103.5043.pdf`*, 2011.
7. Medha Atre. For the DISTINCT clause of SPARQL queries. In *Proceedings of the 25th International Conference Companion on World Wide Web*, pages 7–8, 2016.
8. Franz Baader and Tobias Nipkow. *Term rewriting and all that*. Cambridge university press, 1998.
9. Dimitris Bilidas and Manolis Koubarakis. Efficient duplicate elimination in SPARQL to SQL translation. In *Description Logics*, 2018.
10. Bart Bogaerts, Maxime Jakubowski, and Jan Van den Bussche. Expressiveness of SHACL features and extensions for full equality and disjointness tests. *arXiv preprint `https://arxiv.org/pdf/2212.03553.pdf`*, 2022.
11. Jeen Broekstra, Arjohn Kampman, and Frank Van Harmelen. Sesame: An architecture for storing and querying RDF data and schema information. *Spinning the Semantic Web*, pages 197–222, 2001.
12. Carlos Buil-Aranda, Marcelo Arenas, Oscar Corcho, and Axel Polleres. Federating queries in SPARQL 1.1: Syntax, semantics and evaluation. *Journal of Web Semantics*, 18(1):1–17, 2013.
13. Diego Calvanese, Benjamin Cogrel, Sarah Komla-Ebri, Roman Kontchakov, Davide Lanti, Martin Rezk, Mariano Rodriguez-Muro, and Guohui Xiao. Ontop: Answering SPARQL queries over relational databases. *Semantic Web*, 8(3):471–487, 2017.
14. Upen S Chakravarthy, John Grant, and Jack Minker. Foundations of semantic query optimization for deductive databases. In *Foundations of deductive databases and logic programming*, pages 243–273. Elsevier, 1988.
15. Sijin Cheng and Olaf Hartig. OPT+: A monotonic alternative to OPTIONAL in SPARQL. *Journal of Web Engineering*, 2019.
16. W3C Consortium. Apache jena. `https://jena.apache.org`, 2016.
17. Julien Corman, Fernando Florenzano, Juan L Reutter, and Ognjen Savkovic. SHACL2SPARQL: Validating a SPARQL endpoint against recursive SHACL constraints. In *ISWC (Satellites)*, pages 165–168, 2019.
18. Julien Corman, Fernando Florenzano, Juan L Reutter, and Ognjen Savković. Validating SHACL Constraints over a SPARQL Endpoint. In *International Semantic Web Conference*, pages 145–163. Springer, 2019.
19. Julien Corman, Juan L Reutter, and Ognjen Savković. Semantics and validation of recursive SHACL. In *International Semantic Web Conference*, pages 318–336. Springer, 2018.

20. Floriana Di Pinto, Domenico Lembo, Maurizio Lenzerini, Riccardo Mancini, Antonella Poggi, Riccardo Rosati, Marco Ruzzi, and Domenico Fabio Savo. Optimizing query rewriting in ontology-based data access. In *Proceedings of the 16th International Conference on Extending Database Technology*, pages 561–572, 2013.
21. O Erling. Implementing a SPARQL compliant RDF triple store using a SQLORDBMS. OpenLink Software Virtuoso. 2001.
22. Inc Franz. AllegroGraph. *Compatible Semantic Technologies,* `https://allegrograph.com`, 2017.
23. Paul WPJ Grefen and Peter MG Apers. Integrity control in relational database systems—an overview. *Data & Knowledge Engineering*, 10(2):187–223, 1993.
24. Steve Harris, Andy Seaborne, and Eric Prud'hommeaux. SPARQL 1.1 query language. *W3C recommendation*, 21(10):778, 2013.
25. Matthias Jarke and Jurgen Koch. Query optimization in database systems. *ACM Computing surveys (CsUR)*, 16(2):111–152, 1984.
26. Xun Jian, Yue Wang, Xiayu Lei, Libin Zheng, and Lei Chen. SPARQL rewriting: Towards desired results. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, pages 1979–1993, 2020.
27. David S Johnson and Anthony Klug. Testing containment of conjunctive queries under functional and inclusion dependencies. In *Proceedings of the 1st ACM SIGACT-SIGMOD symposium on Principles of database systems*, pages 164–169, 1982.
28. Haemin Jung and Wooju Kim. Automated conversion from natural language query to SPARQL query. *Journal of Intelligent Information Systems*, 55(3):501–520, 2020.
29. Mark Kaminski and Egor V Kostylev. Beyond well-designed SPARQL. In *19th international conference on database theory (icdt 2016)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2016.
30. Mark Kaminski, Egor V Kostylev, and Bernardo Cuenca Grau. Query nesting, assignment, and aggregation in SPARQL 1.1. *ACM Transactions on Database Systems (TODS)*, 42(3):1–46, 2017.
31. Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. Querix: A natural language interface to query ontologies based on clarification dialogs. In *5th international semantic web conference (ISWC 2006)*, pages 980–981. Citeseer, 2006.
32. Holger Knublauch and Dimitris Kontokostas. Shapes constraint language (SHACL). *W3C Candidate Recommendation*, 11(8), 2017.
33. Roman Kontchakov, Martin Rezk, Mariano Rodriguez-Muro, Guohui Xiao, and Michael Zakharyaschev. Answering SPARQL queries over databases under OWL 2 QL entailment regime. In *International Semantic Web Conference*, pages 552–567. Springer, 2014.
34. Egor V Kostylev, Juan L Reutter, Miguel Romero, and Domagoj Vrgoč. SPARQL with property paths. In *International Semantic Web Conference*, pages 3–18. Springer, 2015.
35. Egor V Kostylev, Juan L Reutter, and Martín Ugarte. CONSTRUCT queries in SPARQL. In *18th International Conference on Database Theory (ICDT 2015)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.
36. Wangchao Le, Songyun Duan, Anastasios Kementsietsidis, Feifei Li, and Min Wang. Rewriting queries on SPARQL views. In *Proceedings of the 20th international conference on World wide web*, pages 655–664, 2011.
37. Frank Manola, Eric Miller, Brian McBride, et al. RDF primer. *W3C recommendation*, 10(1-107):6, 2004.
38. Aisha Mohamed, Ghadeer Abuoda, Abdurrahman Ghanem, Zoi Kaoudi, and Ashraf Aboulnaga. RDFFrames: knowledge graph access for machine learning tools. *The VLDB Journal*, 31(2):321–346, 2022.

39. Harshvardhan J Pandit, Declan O'Sullivan, and Dave Lewis. Test-driven approach towards GDPR compliance. In *International Conference on Semantic Systems*, pages 19–33. Springer, 2019.
40. Paolo Pareti and George Konstantinidis. A review of SHACL: From data validation to schema reasoning for RDF graphs. *Reasoning Web International Summer School*, pages 115–144, 2021.
41. Paolo Pareti, George Konstantinidis, and Fabio Mogavero. Satisfiability and containment of recursive shacl. *Journal of Web Semantics*, 74:100721, 2022.
42. Paolo Pareti, George Konstantinidis, Fabio Mogavero, and Timothy J Norman. SHACL Satisfiability and Containment. In *International Semantic Web Conference*, pages 474–493. Springer, 2020.
43. Jorge Pérez, Marcelo Arenas, and Claudio Gutierrez. Semantics and complexity of SPARQL. *ACM Transactions on Database Systems (TODS)*, 34(3):1–45, 2009.
44. Axel Polleres, Juan Reutter, and E Kostylev. Nested constructs vs. sub-selects in SPARQL. CEUR Workshop Proceedings, 2016.
45. Eric Prud'hommeaux, Jose Emilio Labra Gayo, and Harold Solbrig. Shape expressions: an RDF validation and transformation language. In *Proceedings of the 10th International Conference on Semantic Systems*, pages 32–40, 2014.
46. E Prud'Hommeaux and A Seaborne. SPARQL query language for RDF. W3C recommendation (january 15, 2008), 2011.
47. Kashif Rabbani, Matteo Lissandrini, and Katja Hose. Optimizing SPARQL qeries using shape statistics. 2021.
48. Raymond Reiter. Nonmonotonic reasoning. In *Exploring artificial intelligence*, pages 439–481. Elsevier, 1988.
49. Mariano Rodriguez-Muro and Martin Rezk. Efficient SPARQL-to-SQL with R2RML mappings. *Journal of Web Semantics*, 33:141–169, 2015.
50. Michael Schmidt, Michael Meier, and Georg Lausen. Foundations of SPARQL query optimization. In *Proceedings of the 13th international conference on database theory*, pages 4–33, 2010.
51. Juan F Sequeda and Daniel P Miranker. Ultrawrap: SPARQL execution on relational data. *Journal of Web Semantics*, 22:19–39, 2013.
52. Ruben Taelman, Joachim Van Herwegen, Miel Vander Sande, and Ruben Verborgh. Comunica: a modular SPARQL query engine for the web. In *The Semantic Web–ISWC 2018: 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II 17*, pages 239–255. Springer, 2018.
53. Ratan Bahadur Thapa and Martin Giese. A source-to-target constraint rewriting for direct mapping. In *International Semantic Web Conference*, pages 21–38. Springer, 2021.
54. Ratan Bahadur Thapa and Martin Giese. Mapping relational database constraints to SHACL. In *The Semantic Web–ISWC 2022: 21st International Semantic Web Conference, Virtual Event, October 23–27, 2022, Proceedings*, pages 214–230, 2022.
55. Ratan Bahadur Thapa and Martin Giese. Optimizing SPARQL queries with SHACL (extended version). Research Report 504, Dept. of Informatics, University of Oslo, July 2023.
56. Guohui Xiao, Roman Kontchakov, Benjamin Cogrel, Diego Calvanese, and Elena Botoeva. Efficient Handling of SPARQL optional for OBDA. In *International Semantic Web Conference*, pages 354–373. Springer, 2018.
57. Xiaoyu Yin, Dagmar Gromann, and Sebastian Rudolph. Neural machine translating from natural language to SPARQL. *Future Generation Computer Systems*, 117:510–519, 2021.