Web Semantics: Science, Services and Agents on the World Wide Web 37–38 (2016) 55–74

Contents lists available at ScienceDirect

Web Semantics: Science, Services and Agents on the World Wide Web

journal homepage: www.elsevier.com/locate/websem

Faceted search over RDF-based knowledge graphs☆

Marcelo Arenas[a], Bernardo Cuenca Grau[b], Evgeny Kharlamov[b], Šarūnas Marciuška[b], Dmitriy Zheleznyakov[b,*]

[a] Pontificia Universidad Catolica de Chile, Vicuna Mackenna 4860, Edificio San Agustin, Macul 7820436 Santiago, Chile

[b] University of Oxford, Department of Computer Science, Information Systems Group, Wolfson Building, Parks Road, Oxford OX1 3QD, UK

ARTICLE INFO

Article history:
Received 22 April 2015
Received in revised form
3 September 2015
Accepted 22 December 2015
Available online 31 December 2015

Keywords:
Faceted search
Ontology
OWL 2
RDF
SPARQL
Algorithms

ABSTRACT

Knowledge graphs such as Yago and Freebase have become a powerful asset for enhancing search, and are being intensively used in both academia and industry. Many existing knowledge graphs are either available as Linked Open Data, or they can be exported as RDF datasets enhanced with background knowledge in the form of an OWL 2 ontology. Faceted search is the de facto approach for exploratory search in many online applications, and has been recently proposed as a suitable paradigm for querying RDF repositories. In this paper, we provide rigorous theoretical underpinnings for faceted search in the context of RDF-based knowledge graphs enhanced with OWL 2 ontologies. We identify well-defined fragments of SPARQL that can be naturally captured using faceted search as a query paradigm, and establish the computational complexity of answering such queries. We also study the problem of updating faceted interfaces, which is critical for guiding users in the formulation of meaningful queries during exploratory search. We have implemented our approach in a fully-fledged faceted search system, SemFacet, which we have evaluated over the Yago knowledge graph.

1. Introduction

Knowledge graphs are large collections of interconnected entities enriched with semantic annotations, which have become powerful assets for enhancing search and are now widely used in both academia and industry. Prominent examples of large-scale knowledge graphs include Yago [1], Freebase [2], Google's Knowledge Graph [3], Facebook's Graph Search [4], Microsoft's Satori [5], and Yahoo's Knowledge Graph [6]. Many existing knowledge graphs are either available as Linked Open Data, or they can be exported as RDF datasets [7] enhanced with OWL 2 ontologies [8] capturing the relevant domain background knowledge.

SPARQL [9] has become the standard language for querying RDF data and OWL ontologies, and an increasing number of applications are relying on RDF, OWL 2, and SPARQL for storing, publishing, and querying data; in particular, access to knowledge graphs is often provided by a SPARQL endpoint. Writing SPARQL queries, however, requires some proficiency in the query language and is not well-suited for the majority of users [10,11]. Thus, an important challenge that has attracted a great deal of attention in the Semantic Web community is the development of simple yet powerful query interfaces for non-expert users [12–17]. This challenge becomes even more critical in the context of knowledge graphs such as Yago or Freebase, which are typically oriented towards end-users search.

Faceted search is a prominent approach for querying collections of entities where users can narrow down the search results by progressively applying filters, called *facets* [18]. A facet typically consists of a predicate (e.g., 'gender' or 'occupation' when querying entities about people) and a set of possible string values (e.g., 'female' or 'research'), and entities in the collection are annotated with predicate-value pairs. During faceted search users iteratively select facet values and the entities annotated according to the selection are returned as the search result.

Faceted search in the context of RDF has received significant attention and a number of systems have been developed [19–27]. Furthermore, several such systems have been successfully exploited for performing exploratory search over large knowledge graphs such as Freebase [28].

The theoretical underpinnings of faceted search in the context of RDF and knowledge graphs, however, remain relatively

☆ This research was supported by the Royal Society, the EPSRC projects Score!, DBOnto, and MaSI[3] and the EU FP7 project Optique (n. 318338).

* Corresponding author.
E-mail addresses: marenas@ing.puc.cl (M. Arenas), bernardo.cuenca.grau@cs.ox.ac.uk (B. Cuenca Grau), evgeny.kharlamov@cs.ox.ac.uk (E. Kharlamov), sarunas.marciuska@cs.ox.ac.uk (Š. Marciuška), dmitriy.zheleznyakov@cs.ox.ac.uk (D. Zheleznyakov).

http://dx.doi.org/10.1016/j.websem.2015.12.002
1570-8268/

unexplored [10,29,30]. In particular, the following key questions have not been satisfactorily addressed in the literature (see our Related Work section):

(Q1) What fragments of SPARQL can be naturally captured using faceted search as a query paradigm?
(Q2) What is the complexity of answering such queries?
(Q3) What does it mean to generate and interactively update an interface according to a given RDF graph?

Questions 1 and 2 correspond to the study of the expressive power and complexity of query languages. These are central topics in data management, and addressing them is a key requirement to develop information systems that can provide correctness, robustness, scalability, and extensibility guarantees. Moreover, update (Question 3) is a key task in information systems where query formulation is fundamentally interactive. Our first goal is to answer these questions, thus providing rigorous and solid foundations for faceted search over RDF data.

Our second aim is to provide a framework for faceted search that is also applicable to the wider setting of OWL 2 and hence to ontology-enriched knowledge graphs such as Freebase and Yago. Existing works have focused mostly on RDF, thus essentially disregarding the role of OWL 2 ontologies. We see this as an important limitation. Ontological axioms not only can be used to enrich query answers over RDF datasets with implicit information, but also to enhance the navigation process by providing rich schema-level structure. Furthermore, RDF-based faceted search systems are data-centric and hence cannot be exploited to browse large ontologies such as SNOMED CT [31] or to formulate meaningful queries at the schema level.

More specifically, we formalise in Section 3 our notions of faceted interface and query, which are tailored towards RDF and OWL 2. Our notion of interface enables navigation across interconnected collections of entities, which is inherent to faceted search over RDF data. Furthermore, it abstracts from considerations specific to GUI design (e.g., facet and value ranking), while at the same time reflecting the core functionality of existing systems. Specifically, our interfaces capture both the combination of facets displayed during search and the facet values selected by users. The latter determine a *faceted query*, whose answers constitute the current results of the search. We describe such queries both as first-order logic queries satisfying certain restrictions as well as a fragment of SPARQL.

In Section 4, we study the problem of answering faceted queries over RDF graphs and ontologies captured by the OWL 2 profiles [32]—language fragments with favourable computational properties that are sufficiently powerful to capture the ontologies underpinning most existing knowledge graphs. For each of these profiles we establish tight complexity bounds and propose query answering algorithms.

In Section 5, we focus on interface generation and update. Existing techniques for RDF are based on exploration of the underlying RDF graph. We lift this approach by proposing a graph-based representation of OWL 2 ontologies and their logical entailments for the purpose of faceted navigation, which we refer to as a *facet graph*. Then, we characterise what it means for an interface to conform to an ontology, in the sense that every facet and facet value in the interface is justified by an edge in the graph (and hence by an entailment of the ontology). Finally, we propose generic interface generation and update algorithms that rely on the information in the graph, and show tractability of these tasks for ontologies in the OWL 2 profiles.

In Section 6, we present our faceted search system SemFacet and report on a proof of concept performance evaluation as well as on our practical experience with Yago.

This paper extends our conference publication [33] by providing (i) detailed proofs of our technical results; (ii) a precise account of the connection between our theoretical results in terms of first-order logic and the SPARQL standard; (iii) a detailed description of our system SemFacet; and (iv) a concrete case study based on Yago.[1]

2. Preliminaries

We use standard notions from first-order logic. We assume pairwise disjoint infinite sets of *constants* **C**, *unary predicates* **UP**, and *binary predicates* **BP**. A *signature* is a subset of $\mathbf{C} \cup \mathbf{UP} \cup \mathbf{BP}$. W.l.o.g., we assume all formulae to be rectified, that is, no variable appears free and quantified in a first-order formula φ, and every variable is quantified at most once in φ. The set of free variables of a formula φ is denoted as $\mathsf{fvar}(\varphi)$.

A *fact* is a ground relational atom and a *dataset* is a finite set of facts. A *rule* is a sentence $\forall \mathbf{x} \forall \mathbf{z} [\varphi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})]$, where $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{y}$ are pairwise disjoint variable tuples, the *body* $\varphi(\mathbf{x}, \mathbf{z})$ is a conjunction of atoms with variables in $\mathbf{x} \cup \mathbf{z}$, and the *head* $\exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})$ is an existentially quantified non-empty conjunction of atoms $\psi(\mathbf{x}, \mathbf{y})$ with variables in $\mathbf{x} \cup \mathbf{y}$. Note that we consider only rules that are *Horn* (i.e., disjunction-free), which is sufficient to capture all three profiles of OWL 2. As usual, we assume rules to be *safe*; that is, every universally quantified variable in the rule occurs in a body atom. Universal quantifiers in rules are omitted for brevity. We say that a rule is *Datalog* if its head has at most one atom and all variables are universally quantified. Finally, we define an *ontology* as a finite set of rules and facts. Note that the restriction of rule heads being non-empty ensures satisfiability of any ontology, which makes query results meaningful.

We treat $\top$ as a special symbol in **UP**, which is used to represent a tautology, and assume that any ontology with signature V mentioning $\top$ includes also the following rules:

$$A(x) \rightarrow \top(x) \quad \text{for each } A \in \mathbf{UP} \cap V,$$
$$R(x, y) \rightarrow \top(z) \quad \text{for each } z \in \{x, y\} \text{ and } R \in \mathbf{BP} \cap V.$$

This treatment of $\top$ allows us to ensure safety of rules obtained from OWL 2 ontologies. Similarly, we treat equality $\approx$ as an ordinary predicate in **BP**, and assume that any ontology with signature V mentioning equality contains the following rules axiomatising its meaning:

$$x \approx y \rightarrow y \approx x,$$
$$x \approx y \wedge y \approx z \rightarrow x \approx z,$$
$$R(x, y) \rightarrow z \approx z \quad \text{for all } z \in \{x, y\}, R \in \mathbf{BP} \cap V,$$
$$A(x) \rightarrow x \approx x \quad \text{for all } A \in \mathbf{UP} \cap V,$$
$$A(x) \wedge x \approx y \rightarrow A(y) \quad \text{for all } A \in \mathbf{UP} \cap V,$$
$$R(x, y) \wedge x \approx z \rightarrow R(z, y) \quad \text{for all } R \in \mathbf{BP} \cap V,$$
$$R(x, y) \wedge y \approx z \rightarrow R(x, z) \quad \text{for all } R \in \mathbf{BP} \cap V.$$

OWL 2 defines three *profiles*: weaker languages with favourable computational properties [32]. Each profile ontology can be normalised as rules and facts using the correspondence of OWL 2 and first-order logic and a variant of the structural transformation.[2] An ontology where all rules are of the form given in Table 1 is

- RL if it does not contain rules (3);
- EL if it does not contain rules (1), (9), and (13); and

[1] Some of the material in this paper has also been presented at workshops without formal proceedings [34–36]; a preliminary version of SemFacet was presented as a poster [37] and a short demo paper [38].

[2] Note that the profiles provide the special concept $\bot$, which is immaterial to query answering over satisfiable profile ontologies.

Table 1
Rules corresponding to OWL 2 profiles.

(1) $A(x) \wedge R(x, y_1) \wedge B(y_1) \wedge R(x, y_2) \wedge B(y_2) \rightarrow y_1 \approx y_2$,		(2) $R(x, y) \rightarrow S(x, y)$,
(3) $A(x) \rightarrow \exists y[R(x, y) \wedge B(y)]$,	(4) $A(x) \rightarrow x \approx a$,	(5) $R(x, y) \wedge S(y, z) \rightarrow T(x, z)$,
(6) $A(x) \rightarrow B(x)$,	(7) $A(x) \wedge B(x) \rightarrow C(x)$,	(8) $R(x, y) \rightarrow A(x)$,
(9) $A(x) \wedge R(x, y) \rightarrow B(y)$,	(10) $A(x) \rightarrow R(x, a)$,	(11) $R(x, a) \rightarrow B(x)$,
(12) $R(x, y) \rightarrow A(y)$,	(13) $R(x, y) \rightarrow S(y, x)$,	(14) $R(x, y) \wedge B(y) \rightarrow A(x)$

- QL if it does not contain rules (1), (4), (5), (7), (9), (10), (11), and (14).

Let V be a signature, $\mathsf{at}(V)$ the set of equality-free and constant-free atoms over V, and $\mathsf{eq}(V)$ the set of atoms $x \approx c$ with x a variable and c a constant from V. A *positive existential query* (PEQ) $Q(\mathbf{x})$ is a formula with free variables $\mathbf{x}$, constructed using $\wedge$, $\vee$ and $\exists$ from atoms in $\mathsf{at}(V) \cup \mathsf{eq}(V)$. A PEQ Q is *monadic* if $\mathsf{fvar}(Q)$ is a singleton. It is a *conjunctive query* (CQ) if it is $\vee$-free, and it is a *union of conjunctive queries* (UCQ) if it is of the form $\bigvee_{i=1}^{n} Q_i'(\mathbf{x})$ where each Q_i' is a CQ with the same free variables $\mathbf{x}$ as Q.

We consider two different semantics for query answering. Under the *classical semantics*, a tuple $\mathbf{t}$ of constants is an *answer* to PEQ $Q(\mathbf{x})$ w.r.t. an ontology $\mathcal{O}$ if $\mathcal{O} \models Q(\mathbf{t})$. Under the *active domain semantics*, $\mathbf{t}$ is an answer to Q w.r.t. $\mathcal{O}$ if there is a tuple $\mathbf{t}'$ of constants from $\mathcal{O}$ s.t. $\mathcal{O} \models \varphi(\mathbf{t}, \mathbf{t}')$, where $\varphi(\mathbf{x}, \mathbf{y})$ is the formula obtained from Q by removing all quantifiers.

The evaluation problem under classical (resp. active domain) semantics is to decide, given a tuple of constants $\mathbf{t}$, a PEQ Q and an ontology $\mathcal{O}$ in a language $\mathcal{L}$, whether $\mathbf{t}$ is an answer to Q w.r.t. $\mathcal{O}$ under the given semantics. The classical semantics is the default in first-order logic, whereas active domain is the default semantics of the SPARQL entailment regimes [39]. The latter can be seen as an approximation of the former (an active domain answer is also an answer under classical semantics, but not vice versa). The differences manifest themselves only in the presence of existentially quantified rules and queries; thus, both semantics coincide if either the input ontology is Datalog (and, in particular, if there is no ontology and we consider only RDF data), or if all variables in the input query are free.

3. Faceted interfaces and queries

In this section we provide rigorous logic-based foundations for faceted search over RDF data and OWL 2 ontologies. Specifically, we formalise our notions of *faceted interface* and *faceted query*. Furthermore, we describe faceted queries both in terms of first-order logic and as a fragment of the SPARQL query language. To motivate our definitions we use an example based on an excerpt of DBpedia, where our goal is to find US presidents who graduated from Harvard or Georgetown and have a child who graduated from Stanford.

Example 1. The URIs :*tr* and :*bc* for Theodore Roosevelt and Bill Clinton are annotated with the category 'president'. Roosevelt's son Kermit :*kr* and Clinton's daughter Chelsea :*cc* are categorised as 'person'. Georgetown :*g*, Harvard :*h*, and Stanford :*s* are categorised under 'university', and the USA :*us* and UK :*uk* as 'country'. These annotations are given in RDF and correspond to the following facts:

$$\begin{array}{lll} \mathsf{President}(:tr), & \mathsf{President}(:bc), & \mathsf{Person}(:kr), \\ \mathsf{Person}(:cc), & \mathsf{Country}(:us), & \mathsf{Country}(:uk), \\ \mathsf{Univ}(:h), & \mathsf{Univ}(:g), & \mathsf{Univ}(:s). \end{array}$$

Specific information about entities is represented by literals. For example, Theodore Roosevelt's date of birth is encoded as $\mathsf{dateOfBirth}(:tr, 1858\text{-}10\text{-}27)$. Most importantly, entities are also annotated with other entities; such annotations are given in RDF and correspond to the following facts relating people to their citizenship and to the university they graduated from:

$$\begin{array}{llll} \mathsf{citiz}(:tr, :us), & \mathsf{citiz}(:bc, :us), & \mathsf{child}(:tr, :kr), & \mathsf{child}(:bc, :cc), \\ \mathsf{grad}(:tr, :h), & \mathsf{grad}(:bc, :g), & \mathsf{grad}(:kr, :h), & \mathsf{grad}(:cc, :s). \end{array}$$

Finally, DBpedia can be extended with ontological rules, which describe the meaning of the predicates and constants in the vocabulary. Consider for example the rules given next, which can be captured by the EL profile of OWL 2:

$$\mathsf{President}(x) \wedge \mathsf{citiz}(x, :us) \rightarrow \mathsf{USpres}(x), \tag{1}$$

$$\mathsf{USpres}(x) \rightarrow \mathsf{President}(x) \wedge \mathsf{citiz}(x, :us), \tag{2}$$

$$\mathsf{grad}(x, y) \rightarrow \mathsf{Person}(x) \wedge \mathsf{Univ}(y), \tag{3}$$

$$\mathsf{Person}(x) \rightarrow \exists y\big(\mathsf{citiz}(x, y) \wedge \mathsf{Country}(y)\big). \tag{4}$$

Rules (1) and (2) define US presidents as presidents with US nationality. Rule (3) specifies that the predicate grad relates people to the universities they graduated from. Finally, (4) mandates that each person has a (possibly unspecified) nationality.

Analogously to traditional faceted search, we represent *facets* as pairs of a predicate and a set of values. In the context of RDF, however, entities can be used to annotate other entities, and thus annotations form a graph, rather than a tree. Thus, facet values can be either entity URIs or literals. Examples of facet predicates are the 'graduated from' and 'date of birth' relations, and example values are the URI for Stanford or literals such as Theodore Roosevelt's date of birth. Selection of multiple values within a facet can be interpreted conjunctively or disjunctively, and hence we distinguish between conjunctive and disjunctive facets. We also distinguish a special facet type, whose values are categories (i.e., unary predicates) rather than entities or literals. Finally, a special value any denotes the set of all values compatible with the facet predicate.

Definition 2. Let type and any be symbols not occurring in $\mathbf{C} \cup \mathbf{UP} \cup \mathbf{BP}$. A *facet* is a pair $(X, \circ\Gamma)$, with $\circ \in \{\wedge, \vee\}$, Γ a non-empty set, and either (i) $X = \mathsf{type}$ and $\Gamma \subseteq \mathbf{UP}$, or (ii) $X \in \mathbf{BP}$, $\mathsf{any} \in \Gamma$ and either $\Gamma \subseteq \mathbf{C} \cup \{\mathsf{any}\}$ or $\Gamma \subseteq \mathbf{UP} \cup \{\mathsf{any}\}$. A facet of the form $(X, \wedge\Gamma)$ is *conjunctive*, and a facet of the form $(X, \vee\Gamma)$ is *disjunctive*. In a facet $F = (X, \circ\Gamma)$, X is the *facet predicate*, denoted by $F|_1$, and Γ contains the *facet values* and it is denoted by $F|_2$.

Example 3. The following facets are relevant to our example:

$$F_1 = (\mathsf{type}, \vee\{\mathsf{USpres}, \mathsf{Country}\}),$$
$$F_2 = (\mathsf{child}, \vee\{\mathsf{any}, :kr, :cc\}),$$
$$F_3 = (\mathsf{grad}, \vee\{\mathsf{any}, :h, :s, :g\}),$$
$$F_4 = (\mathsf{citiz}, \wedge\{\mathsf{any}, :us, :uk\}),$$
$$F_5 = (\mathsf{citiz}, \vee\{\mathsf{any}, :us, :uk\}).$$

The disjunctive facet F_1 can be exploited to select the categories to which the relevant entities belong. Facet F_2 can be used to narrow down search results to those individuals with children. In particular, given that F_2 is a disjunctive facet, if the values :*kr* and :*cc* are selected in F_2, then we narrow down the search to those individuals that have Kermit Roosevelt or Chelsea Clinton as children. Furthermore, the value any in F_2 can be used to state that

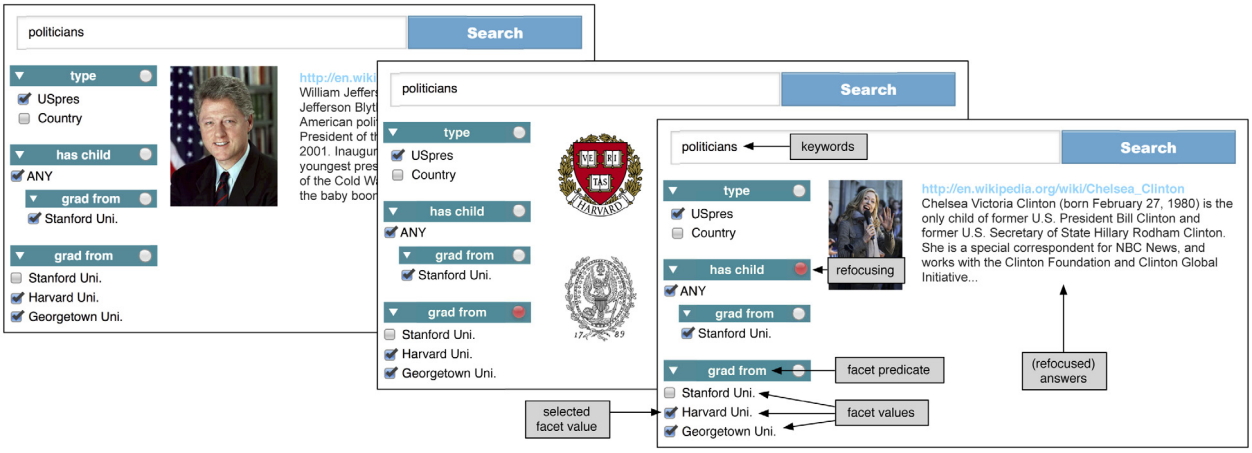


Fig. 1. Left: a visualisation the faceted interface from Example 5 in our SemFacet system; Centre and Right: refocusing of this faceted interface on universities and children of US presidents (as in Example 13).

we are not looking for any specific child. The intuition behind F_3 and F_5 is analogous. Similarly, F_4 is a facet that can also be used to reduce search results. However, if values :*us* and :*uk* are selected in this conjunctive facet, then we narrow down the search to those individuals which are citizens of both the US and the UK.

3.1. The notion of faceted interface

We next move on to the definition of a faceted interface, which encodes a query (the answers to which determine the search results) as well as the choices of facet values available for further refinement.

Definition 4. A *basic faceted interface* (BFI) is a pair (F, Σ), with F a facet and $\Sigma \subseteq F|_2$ the set of *selected values*. The set of *faceted interfaces* (or interfaces, for short) is defined as follows, where I_0 and $I_1 = (F, \Sigma)$ are BFIs and $F|_1 \in$ **BP**:

$$I ::= path \mid (path \wedge path) \mid (path \vee path),$$
$$path ::= I_0 \mid (I_1/I).$$

A BFI encodes user choices for a specific facet, e.g., the BFI $(F_1, \{\mathsf{USpres}\})$ selects the entities categorised as US presidents. BFIs are put together in *paths*: sequences of nested facets that capture navigation between sets of entities annotated with other entities by means of binary relations (e.g., child connects parents to their children); thus, nesting (I_1/I) requires the BFI I_1 to have a binary relation as facet predicate. With nesting we can capture queries such as 'people with a child who graduated from Stanford' by using the interface $(F_2, \{\mathsf{any}\})/(F_3, \{:s\})$ which first selects people having (any) children and then those children with a Stanford degree. Finally, two types of branching can be applied: $(path_1 \wedge path_2)$ indicates that search results must satisfy the conditions specified by both $path_1$ and $path_2$, while $(path_1 \vee path_2)$ indicates that they must satisfy those in $path_1$ or $path_2$.

Example 5. Consider the following interface I_{ex}, which is depicted in our system as on the left-hand side of Fig. 1.

$$\big((F_1, \{\mathsf{USpres}\}) \wedge (F_3, \{:h, :g\})\big) \wedge \big((F_2, \{\mathsf{any}\})/(F_3, \{:s\})\big).$$

The interface consists of three paths connected by $\wedge$-branching. The first path selects US presidents. The second path selects graduates of Harvard or Georgetown. The third path selects individuals with a child who is a Stanford graduate. Since paths are combined conjunctively their constraints apply simultaneously. Thus, we obtain the US presidents who graduated from either Harvard or Georgetown and who have a child who graduated from Stanford.

Our notion of interface abstracts from several considerations that are critical to GUI design. For instance, it is insensitive to the order of BFIs composed by $\wedge$- or $\vee$-branching, as well as to the order of facet values (which are carefully ranked in practice). Furthermore, we model type-facet values as 'flat', whereas in applications categories are organised hierarchically. Although these issues are important from a front-end perspective, they are immaterial to our technical results.

3.2. Faceted queries

The query encoded by the selected values in an interface is formally specified in terms of first-order logic as given next.

Definition 6. Let I be an interface, and let each x_w with $w \in \{0, 1, \ldots, 9\}^*$ be a variable. The query of I is the formula $Q[I] = [\![I, x_\varepsilon, x_0]\!]$ with free variable x_ε defined as in Table 2.

Our semantics assigns to each interface a PEQ with one free variable. For each facet F we have $[\![(F, \emptyset), v, x_w]\!] = \top(v)$, indicating that no restriction is imposed by F if no value is selected. BFIs with a type-facet are interpreted as the conjunction (disjunction) of unary atoms over the same variable. BFIs having as facet predicate a binary predicate result in either an atom whose second argument is existentially quantified (if any is selected), or in a conjunction (disjunction) of binary atoms having a variable as second argument that must be equal to a constant or belong to a unary predicate. Branching $(path_1 \circ path_2)$ with $\circ \in \{\wedge, \vee\}$ is interpreted by constructing the conjunction (disjunction) of the queries for each $path_i$; furthermore, if for some $path_i$ we have that $[\![path_i, v, x_w]\!] = \top(v)$, indicating that no value from the facets occurring in $path_i$ is selected, then $path_i$ is ignored. Finally, nesting involves a 'shift' of variable from the parent BFI to the nested sub-expression.

Example 7. Interface I_{ex} encodes the following query:

$$Q_{\mathsf{ex}}(x) = \mathsf{USpres}(x) \wedge \big(\exists y_1\,(\mathsf{grad}(x, y_1) \wedge y_1 \approx :h) \vee \exists y_2\,(\mathsf{grad}(x, y_2) \wedge y_2 \approx :g)\big) \wedge \exists z\,\big(\mathsf{child}(x, z) \wedge \exists w(\mathsf{grad}(z, w) \wedge w \approx :s)\big).$$

If we consider only facts, the answer set is empty (no entity is categorised as 'US president'). If we also consider the ontology rules, however, we obtain Bill Clinton as the only answer under both classical and active domain semantics.

We can now identify the class of *faceted queries* as the class of first-order queries that can be captured by faceted interfaces.

Table 2
Semantics of faceted interfaces.

Basic Faceted Interfaces If $F = (X, \circ\Gamma)$, then $[\![(F, \Sigma), v, x_w]\!] =$	
$\top(v)$	if $\Sigma = \emptyset$
$\exists x_w\, X(v, x_w)$	if any $\in \Sigma$
$\underset{C \in \Sigma}{\circ} C(v)$	if $X =$ type and $\Sigma \neq \emptyset$
$\underset{t_i \in \Sigma}{\circ} \exists x_{wi}\, X(v, x_{wi}) \wedge x_{wi} \approx t_i$	if $X \neq$ type, any $\notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq$ **C**
$\underset{C_i \in \Sigma}{\circ} \exists x_{wi}\, X(v, x_{wi}) \wedge C_i(x_{wi})$	if $X \neq$ type, any $\notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq$ **UP**
Nesting If $F = (X, \circ\Gamma)$, then $[\![((F, \Sigma)/I), v, x_w]\!] =$	
$\top(v)$	if $\Sigma = \emptyset$
$\exists x_w\, X(v, x_w) \wedge [\![I, x_w, x_{w0}]\!]$	if any $\in \Sigma$
$\underset{t_i \in \Sigma}{\circ} \exists x_{wi}\, X(v, x_{wi}) \wedge x_{wi} \approx t_i \wedge [\![I, x_{wi}, x_{wi0}]\!]$	if any $\notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq$ **C**
$\underset{C_i \in \Sigma}{\circ} \exists x_{wi}\, X(v, x_{wi}) \wedge C_i(x_{wi}) \wedge [\![I, x_{wi}, x_{wi0}]\!]$	if any $\notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq$ **UP**
Branching $[\![(path_1 \circ path_2), v, x_w]\!] =$	
$([\![path_1, v, x_{w0}]\!] \circ [\![path_2, v, x_{w1}]\!])$	if $[\![path_1, v, x_{w0}]\!] \neq \top(v)$ $[\![path_2, v, x_{w1}]\!] \neq \top(v)$
$[\![path_1, v, x_{w0}]\!]$	if $[\![path_1, v, x_{w0}]\!] \neq \top(v)$ $[\![path_2, v, x_{w1}]\!] = \top(v)$
$[\![path_2, v, x_{w1}]\!]$	if $[\![path_1, v, x_{w0}]\!] = \top(v)$ $[\![path_2, v, x_{w1}]\!] \neq \top(v)$
$\top(v)$	if $[\![path_1, v, x_{w0}]\!] = \top(v)$ $[\![path_2, v, x_{w1}]\!] = \top(v)$

Definition 8. A first-order formula φ is a *faceted query* if there exists a faceted interface I such that φ and $Q[I]$ are identical modulo renaming of variables.

3.3. Faceted queries as restricted PEQs

Faceted queries correspond to PEQs of a rather restricted shape, which is determined by Table 2. We next specify such restrictions, which we exploit later on in Section 4 to establish tractability results for query evaluation.

The first observation we can make in Table 2 is that variables in a faceted query can be arranged in a tree with root x_ε and where each variable $x_{w.i}$ is a child of x_w. The tree-shaped nature of faceted queries is captured by the following definition, and we can readily check that query $Q_{\mathsf{ex}}(x)$ in Example 7 is indeed tree-shaped.

Definition 9. Let $Q(x)$ be a monadic PEQ. *The graph* of Q is the smallest directed graph G_Q with a node for each variable in Q and a directed edge (y, y') for each atom $R(y, y')$ occurring in Q where R is different from $\approx$. Moreover, Q is *tree-shaped* if (i) G_Q is a (possibly empty) directed tree rooted at x; (ii) for each edge (y, y') there is at most one binary atom in Q of the form $R(y, y')$.

The second important observation in Table 2 is that disjunction in a faceted query originates from either a disjunctive facet or from $\vee$-branching between paths. In either case, disjunctive subqueries are monadic tree-shaped PEQs.

These observations are reflected in the following proposition.

Proposition 10. *Every faceted query Q is a monadic tree-shaped PEQ with the following property: if $\varphi = (\varphi_1 \vee \varphi_2)$ is a subformula of Q, then* $\mathsf{fvar}(\varphi_1) = \mathsf{fvar}(\varphi_2) = \{x\}$ *for some variable x.*

Proof. The claim in the proposition follows by a simple induction on the structure of faceted queries. We show that for every interface I the query $[\![I, x_\varepsilon, x_0]\!]$ is a monadic tree-shaped PEQ with a single free variable x_ε at the root of the tree and satisfying the property stated in the proposition.

Consider Table 2. For the base case consider BFIs. It can be immediately seen that all queries are monadic PEQs with free variable v. Furthermore, they are tree-shaped with v at the root of the tree and (existentially quantified) variables x_w and $x_{w.i}$ as children of v in the graph of the query.

Let us now consider nesting. The first case is direct. For the remaining cases we know, by the induction hypothesis, that $[\![I, x_w, x_{w0}]\!]$ and $[\![I, x_{wi}, x_{wi0}]\!]$ are monadic tree-shaped PEQs with free variable x_w (resp. x_{wi}) at the root of the tree, and satisfying the property in the proposition. Since variable x_w (resp. x_{wi}) becomes existentially quantified, then $[\![((F, \Sigma)/I), v, x_w]\!]$ has v as a free variable; furthermore, it is tree-shaped with v the new root of the tree. Again, a disjunctive formula is introduced if $\circ$ is $\vee$ and each of the disjuncts has v as common free variable.

The case for branching of paths also follows directly from the inductive hypothesis. □

3.4. Expressing faceted queries in SPARQL

We have shown how faceted queries can be seen in terms of first-order logic as a restricted form of PEQs. In practice, however, we need to specify such queries in SPARQL, as they will be executed over an RDF graph. In this section, we show how faceted queries can be expressed in SPARQL by slightly modifying the transformation rules given in Table 2. We will use an example to explain the main ideas behind this modified transformation and to provide a cleaner picture of the features of SPARQL that are needed to capture faceted queries; the construction sketched by our example can be easily generalised to all the cases given in Table 2. Throughout this section we assume basic familiarity with SPARQL, and refer the reader to the normative documents for further details [9].

Consider the facets defined in Example 3 and the interface I_{ex} in Example 5. To encode the corresponding query in SPARQL, we first need to translate unary and binary relational atoms into SPARQL triple patterns. More precisely, an atom of the form $A(x)$, where x is a variable, is translated into a triple pattern `?x rdf:type :A`, where `?x` is a SPARQL variable representing variable x, `:A` is a URI representing unary predicate A, and `rdf:type` is a reserved URI used to indicate that `?x` is of type of `:A`. Thus, the previous triple pattern asks for all the values for variable x that are elements of A, which is the intended meaning of $A(x)$. Similarly, an atom of the form $R(x, y)$, where x and y are variables, is translated into a triple pattern: `?x :R ?y`, where `:R` is a URI representing the binary predicate R.

Let us consider the interface $(F_1, \{\mathsf{USpres}\})$, which is the first component of I_{ex}, and its corresponding first-order logic query

USpres(x). We capture this query in SPARQL by means of the following query:

```
SELECT ?x
WHERE { ?x rdf:type :USpres . }
```

In this case, we first translate the atom USpres(x) into a triple pattern, and then we indicate that we want to retrieve the value of variable x by using the query form SELECT ?x.

Let us consider the interface $(F_2, \{\mathsf{any}\})$ in our example, whose query is encoded as $\exists y\, \mathsf{child}(x, y)$ in first-order logic. We encode such query in SPARQL as follows:

```
SELECT ?x
WHERE { ?x :child ?y . }
```

As in the previous case, we first translate child(x, y) into a triple pattern, and then we indicate that we want to retrieve all persons who have a child by using the query form SELECT ?x.

Consider the interface $((F_2, \{\mathsf{any}\})/(F_3, \{:s\}))$, which is translated recursively into first-order logic. Following Table 2, we first construct a query of the form $\exists y\, (\mathsf{child}(x, y) \wedge \varphi(y))$ from $(F_2, \{\mathsf{any}\})$, and then replace $\varphi(y)$ by the query encoding $(F_3, \{:s\})$, namely $\exists z(\mathsf{grad}(y, z) \wedge z \approx :s)$. This recursive procedure can be easily adapted to generate a SPARQL query. For this, we first construct a template of the form:

```
SELECT ?x
WHERE { ?x :child ?y.  φ(?y). }
```

and then we recursively invoke the procedure to replace $\varphi(?y)$ by a SPARQL query for the interface $(F_3, \{:s\})$. Finally, the SPARQL query corresponding to $((F_2, \{\mathsf{any}\})/(F_3, \{:s\}))$, which retrieves all persons having a child who graduated from Stanford, is as follows:

```
SELECT ?x
WHERE {
   ?x :child ?y .
   {
      SELECT ?y
      WHERE {
         ?y :grad ?z .
         FILTER (?z = :s) }
   }
}
```

Note that the FILTER operator is used to indicate that the value of variable ?z must be equal to the URI :s. Furthermore, observe that the translation of the interface nesting construct in our language requires the use of nested queries, which were introduced as a new feature in SPARQL 1.1 [9].

So far we have shown four key features of SPARQL needed to encode faceted queries, namely triple patterns to encode unary and binary relational atoms, the query form SELECT to provide the output variable, nested queries to encode interface nesting and the FILTER operator to encode equality atoms. We are only missing one additional feature that is needed for the transformation rules in Table 2: a restricted form of use of the SPARQL operator UNION. Consider the faceted interface $(F_3, \{:h, :g\})$ in our running example. As F_3 is a disjunctive facet, $(F_3, \{:h, :g\})$ is encoded as follows in first-order logic:

$$\exists y_1\, (\mathsf{grad}(x, y_1) \wedge y_1 \approx :h) \vee \exists y_2\, (\mathsf{grad}(x, y_2) \wedge y_2 \approx :g).$$

The two disjuncts of this first-order query are translated into SPARQL as shown before, and are then combined by means of the UNION operator as follows:

```
SELECT ?x
WHERE {
   {
      SELECT ?x
      WHERE {
         ?x :grad ?y1 .
         FILTER (?y1 = :h) }
   }
   UNION
   {
      SELECT ?x
      WHERE {
         ?x :grad ?y2 .
         FILTER (?y2 = :g) }
   }
}
```

Notice that the operator UNION must be used in SPARQL inside a query form, which is why in this case we need to include the outermost query form SELECT ?x. More importantly, for every sub-query of the form P$_1$ UNION P$_2$ it holds that both P$_1$ and P$_2$ have exactly one output variable, which must be the same. This restriction in the use of UNION corresponds to that in Proposition 10 in the context of first-order logic.

3.5. Faceted interfaces with refocusing

The interface in Example 5 finds presidents (such as Bill Clinton) who graduated from either Harvard or Georgetown and have children who graduated from Stanford. If we want to know who these children are (i.e., see Chelsea Clinton as an answer), we must provide *refocusing* (or *pivoting*) functionality [26,27]. We now extend faceted interfaces with such functionality.

Definition 11. Let focus be a symbol not in $\mathbf{C}v \cup \mathbf{UP} \cup \mathbf{BP}$. An *extended basic faceted interface* (EBFI) is either a BFI or a pair $(F, \Sigma \cup \{\mathsf{focus}\})$, where (F, Σ) is a BFI and $F|_1 \in \mathbf{BP}$. Moreover, the set of *extended faceted interfaces* (EFIs) is defined by the same grammar given in Example 5, but where I_0 is a BFI and $I_1 = (F, \Delta)$ is an EBFI with $F|_1 \in \mathbf{BP}$. Finally, each EFI I must have at most one occurrence of the symbol focus.

The value focus is used to change the free variable of the query Q, which determines the kinds of objects returned as answers. Thus, refocusing is used over a facet that introduces new variables in the query, which by Table 2 requires $F|_1 \in \mathbf{BP}$.

The query encoded by an extended interface can be specified in terms of first-order logic as given next.

Definition 12. Let I be an EFI and $[\![I, x_\varepsilon, x_0]\!]$ be a formula defined by the extension of Table 2 with the rules in Table 3. Then the *query* of I is the formula $Q[I]$ defined as follows:

$$Q[I] = \begin{cases} [\![I, x_\varepsilon, x_0]\!] & \text{if focus does not occur in } I, \\ \exists x_\varepsilon\, [\![I, x_\varepsilon, x_0]\!] & \text{otherwise.} \end{cases}$$

A formula φ is an *extended faceted query* if there is an EFI I s.t. φ and $Q[I]$ are identical modulo renaming of variables.

Example 13. For example, consider the following EFI I, which is focused on the children of the US presidents:

$$\big((F_1, \{\mathsf{USpres}\}) \wedge (F_3, \{:h, :g\})\big) \wedge \big((F_2, \{\mathsf{focus}\})/(F_3, \{:s\})\big).$$

Then, $Q[I]$ is obtained from $Q_{\mathsf{ex}}(x)$ in Example 7 by first dropping the existential quantifier $\exists z$ from $Q_{\mathsf{ex}}(x)$, and then adding $\exists x$ to the

Table 3
Semantics of extended faceted interfaces.

Extended Basic Faceted Interfaces	
If $F = (X, \circ\Gamma)$, then $[\![(F, \Sigma \cup \{\text{focus}\}), v, x_w]\!] =$	
$X(v, x_w)$	if $\Sigma = \emptyset$
$[\![(F, \{\text{focus}\}), v, x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{C} \cup \{\text{any}\}$
$[\![\big((F, \{\text{focus}\})/((\text{type}, \vee\Gamma), \Sigma)\big), v, x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{UP} \cup \{\text{any}\}$
Nesting	
If $F = (X, \circ\Gamma)$, then $[\![((F, \Sigma \cup \{\text{focus}\})/I), v, x_w]\!] =$	
$X(v, x_w) \wedge [\![I, x_w, x_{w0}]\!]$	if $\Sigma = \emptyset$
$[\![((F, \{\text{focus}\})/I), v, x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{C} \cup \{\text{any}\}$
$[\![\big((F, \{\text{focus}\})/\big(((\text{type}, \vee\Gamma), \Sigma) \wedge I\big)\big), v, x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{UP} \cup \{\text{any}\}$

resulting query, thus obtaining $Q'_{\text{ex}}(z)$:

$$\exists x \big(\mathsf{USpres}(x) \wedge \big(\exists y_1\,(\mathsf{grad}(x, y_1) \wedge y_1 \approx :h) \vee \exists y_2\,(\mathsf{grad}(x, y_2) \wedge y_2 \approx :g)\big) \wedge \big(\mathsf{child}(x, z) \wedge \exists w(\mathsf{grad}(z, w) \wedge w \approx :s)\big)\big).$$

The answer to $Q_{\text{ex}}(z)$ is precisely Chelsea Clinton.

We conclude this section by pointing out that PEQs obtained from faceted interfaces extended with refocusing also satisfy Proposition 10, with the only difference that the corresponding query graph is no longer rooted in the answer variable. Consequently, as we will see later on, refocusing does not increase the complexity of query evaluation.

4. Answering faceted queries

A faceted search system must compute the answers to a query each time that a user selects a facet value to refine the search results. Thus, query evaluation is a key reasoning problem for the development of efficient and robust faceted search systems.

As discussed in Section 3, faceted queries are monadic positive existential queries resulting from the selection of facet values in an interface. By standard results for relational databases, PEQ evaluation is an NP-hard problem, even if we restrict ourselves to CQs and ontologies consisting of just a dataset.

In this section we show that, in contrast to PEQs (and even CQs), faceted query evaluation over datasets is tractable due to the restrictions in the structure of queries imposed by Proposition 10. Furthermore, the problem remains tractable in most cases if we consider ontologies in the OWL 2 profiles. Our tractability results concern *combined complexity*, which takes into account the size of the entire input (i.e., ontological rules, RDF data and queries).

4.1. Faceted query answering over datasets

We next show how the restricted shape of faceted queries can be exploited to make query answering more efficient under both classical and active domain semantics. We start by providing a polynomial time algorithm for answering faceted queries over datasets.[3] The key observation is that the disjunctive subqueries $\varphi = \varphi_1 \vee \varphi_2$ in the input query Q can be evaluated w.r.t. the input data in a bottom-up fashion. To answer one such φ, we solve φ_1 and φ_2 independently and store the answers as facts in the dataset using a fresh unary predicate C_φ associated to φ.

Example 14. Query Q_{ex} in Example 7 can be answered over the dataset in our running example as follows. First, solve the subquery φ asking for graduates from either Harvard or Georgetown; each disjunct is a tree-shaped CQ, and we obtain Bill Clinton, Theodore Roosevelt and Kermit Roosevelt as answers. Then, extend the dataset with facts $C_\varphi(:bc)$, $C_\varphi(:tr)$ and $C_\varphi(:kr)$ over a fresh predicate C_φ. Finally, rewrite Q_{ex} by replacing $\varphi(x)$ with $C_\varphi(x)$ and answer the rewritten query over the extended dataset. We obtain the empty set of answers since no entity is explicitly categorised as US president.

Algorithm 1: ANSWER-FQ

INPUT : $\mathcal{D}$ a dataset; Q a faceted query
OUTPUT: Answers to Q w.r.t. $\mathcal{D}$

1 $S :=$ Set of disjunctive subformulas of Q
2 $\preceq :=$ partial order on S s.t. $\varphi \preceq \varphi'$ iff φ is a subformula of φ'
3 **for each** $\varphi = (\varphi_1 \vee \varphi_2) \in S$ *listed in ascending* $\preceq$*-order* **do**
4 **for each** $1 \leq i \leq 2$ **do**
5 $\varphi'_i :=$ REWRITE(φ_i)
6 $Ans_i :=$ ANSWER-TREE-CQ(φ'_i, $\mathcal{D}$)
7 $\mathcal{D} := \mathcal{D} \cup \{C_{\varphi_1 \vee \varphi_2}(d) \mid d \in Ans_1 \cup Ans_2\}$
8 $Q' :=$ REWRITE(Q)
9 $Ans :=$ ANSWER-TREE-CQ(Q', $\mathcal{D}$)
10 **return** Ans

Function REWRITE

INPUT : φ a faceted query
OUTPUT: A conjunctive query

1 **case** φ *an atom* **return** φ
2 **case** $\varphi = \exists z\, \varphi'$ **return** $\exists z$ REWRITE(φ')
3 **case** $\varphi = \varphi_1 \wedge \varphi_2$ **return** REWRITEx(φ_1) $\wedge$ REWRITE(φ_2)
4 **case** $\varphi = \varphi_1 \vee \varphi_2$ **return** $C_{\varphi_1 \vee \varphi_2}(y)$ with $y = \mathsf{fvar}(\varphi_i)$

Algorithm 1 implements these ideas. The algorithm relies on a specialised algorithm ANSWER-TREE-CQ to answer (monadic) tree-shaped CQs, which is used as a 'black box'. The following theorem establishes correctness of our algorithm.

Theorem 15. *Algorithm* 1 *computes all answers to* Q *w.r.t.* $\mathcal{D}$.

Proof. First, note that the properties of faceted queries given in Proposition 10 and the definition of the function REWRITE ensure that the input passed to ANSWER-TREE-CQ in each call is indeed a tree-shaped conjunctive query.

Correctness of the algorithm follows directly from the following property, which holds in each iteration of the main loop.

($\star$) Let $\varphi = (\varphi_1 \vee \varphi_2) \in S$ be as in Line 3. Then, the answers to φ w.r.t. the input ontology are precisely $Ans_1 \cup Ans_2$ as in Line 7.

In what follows, we show that ($\star$) indeed holds. Consider the case where $\varphi = \varphi_1 \vee \varphi_2$ is $\preceq$-minimal. Then, neither φ_1 nor φ_2 are disjunctive. In this case, φ'_i in Line 5 is precisely φ_i, and Property ($\star$) holds directly by the semantics of first-order logic and the fact that datasets have a single minimal model: d is an answer to φ iff it is an answer to either of its disjuncts.

Consider the case $\varphi = \varphi_1 \vee \varphi_2$ is not $\preceq$-minimal. For each φ_i we have two possibilities: (i) φ_i is not disjunctive, in which case φ'_i in

[3] Note that both semantics coincide in this case.

Algorithm 2: ANSWER-FQ-ACTIVE

INPUT : $\mathcal{O}$ an ontology; Q a faceted query
OUTPUT : Active domain answers to Q w.r.t. $\mathcal{O}$

1 $\mathcal{D}$:= COMPUTE-ENTAILED-FACTS($\mathcal{O}$)
2 *Ans* := ANSWER-FQ(Q, $\mathcal{D}$)
3 **return** *Ans*

Line 5 is precisely φ_i and thus the answers to φ_i' coincide with the answers to φ_i; (ii) φ_i contains disjunctive sub-formulas, in which case the definition of REWRITE ensures that φ_i will be rewritten as disjunction-free by replacing each $\preceq$-maximal disjunctive sub-formula γ of φ_i with $C_\gamma(y)$. But then, since each such $\gamma \preceq \varphi_i$ we have that the modified dataset includes the answers to γ as facts over C_γ. □

Thus, faceted queries can be evaluated in polynomial time with an oracle for the evaluation of tree-shaped CQs. By a classic result in database theory, acyclic CQs (and hence also tree-shaped CQs as in Definition 9) can be answered in polynomial time [40]. Thus, tractability of tree-shaped CQ evaluation transfers to the evaluation of faceted queries.

Corollary 16. *Faceted query evaluation over datasets is feasible in polynomial time.*

In what follows we study query answering over ontologies (and not just datasets) under both active domain and classical semantics.

4.2. Active domain semantics

In practice, queries over ontology-enhanced RDF data are typically represented in SPARQL and executed using off-the-shelf reasoning engines with SPARQL support. The specification of SPARQL under entailment regimes [39] is based on active domain semantics, which requires existentially quantified variables in the query Q to map to actual constants in the input ontology $\mathcal{O}$. In this case, we can answer queries using Algorithm 2, which computes the dataset $\mathcal{D}$ of all facts entailed by $\mathcal{O}$ and then answers Q w.r.t. $\mathcal{D}$. The correctness of Algorithm 2 follows from Theorem 15 and the following lemma.

Lemma 17. *Let Q be a PEQ, let $\mathcal{O}$ be an ontology, and let $\mathcal{D}$ be the set of all facts α such that $\mathcal{O} \models \alpha$. Then, the answer sets to Q w.r.t. $\mathcal{O}$ and w.r.t. $\mathcal{D}$ coincide under active domain semantics.*

Proof. First, note that since $\mathcal{O} \models \mathcal{D}$ and $\mathcal{D}$ is a dataset, every answer to Q w.r.t. $\mathcal{D}$ is an answer to Q w.r.t. $\mathcal{O}$. To show the converse, pick an active domain answer to Q w.r.t. $\mathcal{O}$. By the definition of active domain semantics, there must exist a tuple $\mathbf{t}'$ of constants from $\mathcal{O}$ such that $\mathcal{O} \models \varphi(c, \mathbf{t}')$, where φ is the formula obtained from Q by removing all quantifiers. Clearly, $\varphi(c, \mathbf{t}')$ is a Boolean combination of facts. Since we consider only Horn rules in this paper, we can transform $\mathcal{O}$ into a Logic Program $\mathcal{P}_\mathcal{O}$ by Skolemising existentially quantified variable in rules using functional terms (note that standard Skolemisation preserves entailment). Program $\mathcal{P}_\mathcal{O}$ has a (possibly infinite) Herbrand model $\mathcal{H}$ that can be homomorphically embedded into any other Herbrand model of $\mathcal{P}_\mathcal{O}$ [41]. Furthermore, $\mathcal{H}$ coincides with $\mathcal{D}$ when restricted to constants. We have that $\mathcal{O} \models \varphi(c, \mathbf{t}')$ iff $\mathcal{P}_\mathcal{O} \models \varphi(c, \mathbf{t}')$ iff $\mathcal{H} \models \varphi(c, \mathbf{t}')$ iff $\mathcal{D} \models \varphi(c, \mathbf{t}')$. Hence, we can conclude that c is also an answer to Q w.r.t. $\mathcal{D}$. □

By showing that fact entailment is tractable for all the profiles, we can immediately prove tractability of faceted query evaluation under active domain semantics. Thus, by committing to the active domain semantics of SPARQL we achieve tractability without emasculating the ontology language.

Theorem 18. *Active domain evaluation of faceted queries is in* PTIME *w.r.t. all normative OWL 2 profiles. Furthermore, it is* PTIME*-complete w.r.t. the EL and RL profiles.*

Proof. PTIME-hardness for EL and RL follows from the known hardness result for fact entailment in these profiles [32]. We next show membership in PTIME for all profiles. By Lemma 17, it suffices to show tractability of fact entailment. We first observe that entailment of unary facts is feasible in polynomial time since instance checking for atomic class expressions is tractable for each of the profiles [32].

We now argue that checking $\mathcal{O} \models \alpha$ with α a binary fact of the form $R(c, d)$ is also tractable. If $\mathcal{O}$ is an OWL 2 EL ontology, then this is the case iff the following holds, where A is a fresh unary predicate

$$\mathcal{O} \cup \{R(x, d) \rightarrow A(x)\} \models A(c)$$

which can be checked in polynomial time. If $\mathcal{O}$ is in OWL 2 RL, then $\mathcal{O} \models R(c, d)$ iff $R(c, d)$ the fact holds in the Least Herbrand Model of $\mathcal{O}$, which can be computed in polynomial time given that $\mathcal{O}$ has at most three variables per rule. Finally, if $\mathcal{O}$ is in OWL 2 QL, then $\mathcal{O} \models R(c, d)$ iff $\mathcal{O}_P$ does, where $\mathcal{O}_P$ is the subset of facts and rules of Type (2), (10), (12), and (16) (from Table 1) in $\mathcal{O}$. Since $\mathcal{O}_P$ is also an OWL 2 RL ontology then the check is also feasible in polynomial time. □

4.3. Classical semantics

Classical and active domain semantics coincide if we restrict ourselves to Datalog ontologies. Thus, Algorithm 2 can also be used for faceted query answering under classical semantics if the input ontology is Datalog. Since OWL 2 RL ontologies are Datalog it follows that our results in Theorem 18 transfer to OWL 2 RL ontologies under classical semantics.

In contrast to RL, the EL and QL profiles can capture existentially quantified knowledge and hence active domain and classical semantics may diverge for queries with existentially quantified variables.

To deal with EL ontologies, we exploit techniques developed for the *combined approach* to CQ answering [42–44]. As a first step, we rewrite rules of Type (3) in Table 1 into Datalog by Skolemising existentially quantified variables into constants.

Definition 19. Let $\mathcal{O}$ be in EL. The ontology $\Xi(\mathcal{O})$ is obtained from $\mathcal{O}$ by replacing each rule $A(x) \rightarrow \exists y[R(x, y) \wedge B(y)]$ with rules $A(x) \rightarrow P(x, c_{R,B})$, $P(x, y) \rightarrow R(x, y)$, and $P(x, y) \rightarrow B(y)$, where P is a fresh predicate and $c_{R,B}$ is a globally fresh constant uniquely associated with R and B.

Although this transformation strengthens the ontology, it preserves the entailment of all facts [42,45].

Lemma 20 (*Implicit in [44,45]*)**.** *Let $\mathcal{O}$ be an EL ontology and let α be a fact mentioning only constants and predicates from $\mathcal{O}$. Then, $\Xi(\mathcal{O}) \models \alpha$ implies $\mathcal{O} \models \alpha$.*

As we show in the following lemma, this result extends to monadic tree-shaped CQs.

Lemma 21. *Let $\mathcal{O}$ be an EL ontology, let c be a constant from $\mathcal{O}$, and $Q(x)$ a monadic tree-shaped CQ. Then, $\Xi(\mathcal{O}) \models Q(c)$ implies $\mathcal{O} \models Q(c)$.*

Proof. For each constant a in $\mathcal{O}$, let A_a be a fresh unary predicate associated to a. Let $\mathcal{O}_1$ be obtained from $\mathcal{O}$ by adding the fact $A_a(a)$ for each constant a in $\mathcal{O}$. Also, let Q_1 be the CQ obtained from Q by replacing each equality atom $y \approx a$ in Q with $A_a(y)$. It is routine to show that the following holds:

1. The answers to Q w.r.t. $\mathcal{O}$ coincide with the answers to Q_1 w.r.t. $\mathcal{O}_1$;
2. The answers to Q w.r.t. $\Xi(\mathcal{O})$ coincide with the answers to Q_1 w.r.t. $\Xi(\mathcal{O}_1)$.

Consider the Datalog rule $\varphi(x, \mathbf{y}) \rightarrow A_{Q_1}(x)$, where A_{Q_1} is fresh and $\varphi(x, \mathbf{y})$ is the conjunction of atoms in Q_1. Since Q_1 is tree-shaped, then the given rule can be written as the ontology $\mathcal{O}_2$ which we define next. Let G_{Q_1} be the tree associated to Q_1 (c.f. Definition 9), and let P_z be a fresh unary predicate for each variable z in G_{Q_1}. If z is a leaf of G_{Q_1}, let $\mathcal{O}_z$ be as follows:

$$\mathcal{O}_z = \left\{ \bigwedge_{A_i(z) \text{ in } Q_1} A_i(z) \rightarrow P_z(z) \right\}.$$

If z is not a leaf, then $\mathcal{O}_z$ is defined as follows, where $z_1, \ldots, z_n$ are the children of z in G_{Q_1} and $R_j(z, z_j)$ is the unique binary atom involving z and z_j in Q_1:

$$\mathcal{O}_z = \left\{ \bigwedge_{A_i(z) \text{ in } Q_1} A_i(z) \wedge \bigwedge_{j=1}^{n} [R_j(z, z_j) \wedge P_{z_j}(z_j)] \rightarrow P_z(z) \right\}.$$

Then, $\mathcal{O}_2$ is defined as follows:

$$\mathcal{O}_2 = \left[\bigcup_{z \text{ in } Q_1} \mathcal{O}_z \right] \cup \left\{ P_x(x) \wedge \bigwedge_{A_i(x) \text{ in } Q_1} A_i(x) \rightarrow A_{Q_1}(x) \right\}.$$

Clearly, the ontology $\mathcal{O}_2$ can be normalised into both EL and RL. Furthermore, the following holds:

3. The answers to Q_1 w.r.t. $\mathcal{O}_1$ and the instances of A_{Q_1} w.r.t. $\mathcal{O}_1 \cup \mathcal{O}_2$ coincide.
4. The answers to Q_1 w.r.t. $\Xi(\mathcal{O}_1)$ and the instances of A_{Q_1} w.r.t. $\Xi(\mathcal{O}_1 \cup \mathcal{O}_2)$ coincide.

Assume that $\Xi(\mathcal{O}) \models Q(c)$. Then, by Property 2 be have $\Xi(\mathcal{O}_1) \models Q_1(c)$ and by Property 4 $\Xi(\mathcal{O}_1 \cup \mathcal{O}_2) \models A_{Q_1}(c)$. But then, Lemma 20 gives us $\mathcal{O}_1 \cup \mathcal{O}_2 \models A_{Q_1}(c)$. Thus, by Properties 3 and 1 we obtain $\mathcal{O} \models Q(c)$, as required. □

Using Lemma 21, we can show that the evaluation of faceted queries w.r.t. EL ontologies is also preserved under Ξ.

Lemma 22. *Let Q be a faceted query, $\mathcal{O}$ an EL ontology, and let c be a constant in $\mathcal{O}$. Then, $\mathcal{O} \models Q(c)$ iff $\Xi(\mathcal{O}) \models Q(c)$.*

Proof. The left-to-right implication is trivial since $\Xi(\mathcal{O}) \models \mathcal{O}$.

Assume now that $\Xi(\mathcal{O}) \models Q(c)$. Since $Q(c)$ is a PEQ, there is a (maybe exponentially larger) UCQ $U(c) = \bigvee_{i=1}^{n} Q_i'(c)$ that is logically equivalent to $Q(c)$. Consequently, $\Xi(\mathcal{O}) \models Q(c)$ iff $\Xi(\mathcal{O}) \models U(c)$. Since $\Xi(\mathcal{O})$ is a Datalog ontology, we have that $\Xi(\mathcal{O}) \models U(c)$ iff $\Xi(\mathcal{O})$ entails some CQ $Q_i'(c)$ occurring as a disjunct in $U(c)$. Hence, it suffices to show that $\mathcal{O} \models Q_i'(c)$. Since $Q(c)$ is tree-shaped, so is $U(c)$ (DNF normalisation does not affect the arrangement of variables), and thus so is $Q_i'(c)$. By Lemma 21 we have that $\mathcal{O} \models Q_i'(c)$, as required. □

It follows that faceted queries over an EL ontology $\mathcal{O}$ can be answered under classical semantics by applying Algorithm 2 to $\Xi(\mathcal{O})$. Since Ξ is a linear transformation and $\Xi(\mathcal{O})$ is an RL ontology, tractability of faceted query evaluation follows.[4]

Theorem 23. *Faceted query evaluation under classical semantics is* PTIME*-complete for RL ontologies and EL ontologies.*

Proof. As in the case of active domain semantics, hardness follows from the known hardness result for fact entailment in these profiles. Since each RL ontology $\mathcal{O}$ is a Datalog program, classical and active domain semantics coincide; hence, we can use Algorithm 2 to evaluate faceted queries under classical semantics as well. Program $\mathcal{O}$ contains at most 3 variables per rule and hence procedure COMPUTE-ENTAILED-FACTS can be implemented in polynomial time. Corollary 16 ensures that ANSWER-FQ is feasible in polynomial time over datasets. In the case of EL, Lemma 22 ensures that we can apply Algorithm 2 to $\Xi(\mathcal{O})$. Since $\Xi(\mathcal{O})$ is RL and can be constructed in linear time, tractability for RL implies tractability for EL. □

In contrast, the evaluation of acyclic CQs is already NP-hard for OWL 2 QL [47] and the proof in [47] can be adapted to also show NP-hardness of faceted query evaluation. Furthermore, we can also show membership in NP, and hence NP-completeness of faceted query evaluation for OWL 2 QL.

Theorem 24. *Faceted query evaluation under classical semantics is* NP*-complete for QL ontologies.*

Proof. We first prove membership in NP. We say that faceted query Q_1 is more specific than Q_2 if Q_1 can be obtained from Q_2 by replacing a subformula $(\varphi_1 \vee \varphi_2)$ of Q_2 by either φ_1 or φ_2. Moreover, we define $\trianglelefteq$ as the reflexive and transitive closure of the relation of being more specific, and given a faceted query Q, we define the determinisation of Q, denoted by $\mathsf{det}(Q)$, as the set of all CQs Q' such that $Q' \trianglelefteq Q$. Determinisation satisfies the following property $(\star)$.

$(\star)$ For every faceted query Q, QL ontology $\mathcal{Q}$ and constant c, it holds that $\mathcal{O} \models Q(c)$ if and only if there exists $Q' \in \mathsf{det}(Q)$ such that $\mathcal{O} \models Q'(c)$.

It is well-known that evaluation of arbitrary CQs is in NP for QL ontologies. From this and $(\star)$ we obtain that faceted query evaluation under classical semantics is in NP for QL ontologies.

We show hardness by adapting the proof of Theorem 1 in [47], which shows NP-hardness of CQ evaluation w.r.t. OWL 2 QL ontologies by reduction from propositional satisfiability. Consider a propositional formula in CNF $\alpha = \bigwedge_{j=1}^{m} D_j$ over variables $p_1, \ldots, p_n$ where each D_j is a propositional clause. Next, consider the following OWL 2 QL ontology $\mathcal{O}$ consisting of the following axioms for $i \in [1, n]$, $j \in [1, m]$ and $k = 0, 1$:

$$\begin{aligned}
C_j(x) &\rightarrow A_0(x), \\
C_j(x) &\rightarrow A_i(x), \\
X_i^k(x) &\rightarrow A_i(x), \\
A_i(x) &\rightarrow \exists y(R(x, y) \wedge A_{i-1}(y)), \\
A_{i-1}(x) &\rightarrow \exists y(S(x, y) \wedge X_i^k(y)), \\
S(x, y) &\rightarrow R(y, x), \\
X_i^0(x) &\rightarrow \exists y(R(x, y) \wedge C_j(y)) \quad \text{if } \neg p_i \in D_j, \\
X_i^1(x) &\rightarrow \exists y(R(x, y) \wedge C_j(y)) \quad \text{if } p_i \in D_j, \\
C_j(x) &\rightarrow \exists y(R(x, y) \wedge C_j(y)), \\
A_0(a).
\end{aligned}$$

Further, when writing faceted interfaces, we will omit sets of selected values for simplicity, that is, we will write $(X, \circ\Gamma)$ instead of $((X, \circ\Gamma), \Sigma)$, assuming that $\Sigma = \Gamma$. Moreover, (X, v) will designate the facet $(X, \vee\{v\})$. Consider now the following family of sub-interfaces for $j \in [1, m]$.

$$\begin{aligned}
E_j = (R, \mathsf{any})/\Big(&(\mathsf{type}, A_{n-1}) \wedge \\
&\big((R, \mathsf{any})/\big((\mathsf{type}, A_{n-2}) \wedge \ldots \\
&\wedge ((R, \mathsf{any})/(\mathsf{type}, \wedge\{A_0, C_j\}))\big)\big)\Big).
\end{aligned}$$

[4] This result is consistent with existing results for acyclic CQs in EL [46].

Next consider the following faceted interface I:

$$I = (\mathsf{type}, A_0) \wedge \Big((S, \mathsf{any}) / \Big((\mathsf{type}, A_1) \wedge \ldots \big((S, \mathsf{any}) \\ / ((\mathsf{type}, A_n) \wedge E_1 \wedge \ldots \wedge E_n) \big) \Big) \Big).$$

Furthermore, $Q[I]$ is isomorphic to the following query, where $\mathbf{y} = (y_1, \ldots, y_n)$ and $\mathbf{z}^j = (z_0^j, \ldots, z_{n-1}^j)$ for $j \in [1, m]$.

$$\exists \mathbf{y} \exists \mathbf{z}^1 \ldots \exists \mathbf{z}^m \Big(A_0(y_0) \wedge \bigwedge_{i=1}^{n} S(y_{i-1}, y_i) \wedge A_i(y_i) \\ \wedge \bigwedge_{j=1}^{m} \Big[R(y_n, z_{n-1}^j) \wedge A_0(z_0^j) \wedge C(z_0^j) \wedge \\ \bigwedge_{i=n-1}^{1} A_i(z_i^j) \wedge R(z_i^j, z_{i-1}^j) \Big] \Big).$$

It can be checked that a is an answer to $Q[I]$ w.r.t. $\mathcal{O}$ iff the propositional formula α is satisfiable. □

4.4. Extended faceted queries

We conclude by arguing that the refocusing functionality does not increase complexity of query evaluation. PEQs obtained from EFIs satisfy Proposition 10, with the only difference that the corresponding query graph is no longer rooted in the answer variable. Algorithm 1 can be extended to prove that Corollary 16 also holds for extended faceted queries. From this, and using the same techniques as in the proofs of Theorems 18 and 24, we obtain the following result.

Theorem 25. *Extended faceted query evaluation under classical semantics is* (i) PTIME*-complete for RL and EL; and* (ii) NP*-complete for QL.*

Moreover, active domain evaluation of extended faceted queries is in PTIME *w.r.t. all normative OWL 2 profiles, and it is* PTIME*-complete for RL and EL.*

Proof. Note that the complexity results we have obtained for faceted queries apply to the class of PEQs satisfying the properties given in Proposition 10 as we did not make in our proofs any further assumptions about the structure of faceted queries.

Let us now consider extended faceted queries and their semantics as in Definition 12. Their structure is exactly the same as regular faceted queries with the only difference that the answer variable does not need to be rooted in variable x_ε. Suppose the answer variable to such query Q is y. To check whether some constant c is an answer to Q we simply add the equality atom $y \approx c$ to Q and existentially quantify y. The result is a Boolean query that is tree-shaped (if we take x_ε as root) and which satisfies the property stated in Proposition 10 for disjunctive subformulas. Hence, the complexity of faceted query evaluation is exactly the same as the complexity of evaluating extended faceted queries. □

5. Interface generation & update

Faceted navigation is an interactive process. Starting with an initial interface generated from a keyword search, users select or unselect facet values and the system reacts to these user actions by updating the search results (query answers) as well as the facets available for further navigation.

Example 26. Consider the interactive construction of our interface I_{ex} from Example 5. Navigation starts with an interface with no selected value, which may have been generated as a response to a keyword search (facets F_i are given in Example 3):

$$I_0 = (F_1, \emptyset) \wedge (F_3, \emptyset) \wedge (F_2, \emptyset) \wedge (F_5, \emptyset).$$

We may then select the category USpres in F_1, which narrows down the search to US presidents. In response, the system may construct the following new interface I_1:

$$I_1 = (F_1, \{\mathsf{USpres}\}) \wedge (F_3, \emptyset) \wedge (F_2, \emptyset).$$

Interface I_1 incorporates the required filter on US presidents. Furthermore, it no longer includes facet F_5 since US presidents have only US nationality and hence any filter over this facet becomes redundant. Next, we select Harvard and Georgetown in facet F_3, which narrows down the search to US presidents with either a Harvard or Georgetown degree and yields the following interface:

$$I_2 = (F_1, \{\mathsf{USpres}\}) \wedge (F_3, \{:h, :g\}) \wedge (F_2, \emptyset).$$

Next, we select any in facet F_2 to look for presidents with children. In response, the system constructs the following interface:

$$I_3 = (F_1, \{\mathsf{USpres}\}) \wedge (F_3, \{:h, :g\}) \wedge \big((F_2, \{\mathsf{any}\}) / (F_3, \emptyset) \big).$$

Interface I_3 provides a nested BFI $(F_3, \emptyset)$, which allows us to select the university that children of US presidents attended. We pick Stanford, and the system finally constructs I_{ex}.

We next propose interface generation and update algorithms that are guided by the (explicit and implicit) information in $\mathcal{O}$. Our algorithms are based on the same unifying principle: each element of the initial interface (resp. each change in response to an action) must be 'justified' by an entailment in $\mathcal{O}$. In this way, by exploring the ontology, we guide users in the formulation of meaningful queries.

There is an inherent degree of non-determinism in faceted navigation: if a user selects a facet value, it is unclear whether the next facet generated by the system should be conjunctive or disjunctive, and whether it should be incorporated in the interface by means of conjunctive or disjunctive branching. In many applications, however, different values in a facet are interpreted disjunctively, whereas constraints imposed by different facets are interpreted conjunctively. Thus, to resolve such ambiguities and devise fully deterministic algorithms, we focus on a restricted class of interfaces where conjunctive facets and disjunctive branching are disallowed.

Definition 27. A faceted interface I is *simple* if all facets occurring in I are disjunctive, and it does not contain sub-interfaces of the form $(path_1 \vee path_2)$.

5.1. The ontology facet graph

We capture the facets that are relevant to an ontology $\mathcal{O}$ in a *facet graph*, which can be seen as a concise representation of $\mathcal{O}$. Our interface generation and update algorithms are parameterised by such graph rather than by $\mathcal{O}$ itself.

The nodes of a facet graph are possible facet values (unary predicates and constants), and edges are labelled with possible facet predicates (binary predicates and type). The key property of a facet graph is that every X-labelled edge (v, w) is justified by a rule or fact entailed by $\mathcal{O}$ which semantically relates v to w via X. We distinguish three kinds of semantic relations: *existential*, where X is a binary predicate and (each instance of) v must be X-related to (an instance of) w in the models of $\mathcal{O}$; *universal*, where (each instance of) v is X-related only to (instances of) w in the models of $\mathcal{O}$; and *typing* where X is type and constant v is entailed to be an instance of the unary predicate w.

Definition 28. A *facet graph* for $\mathcal{O}$ is a directed labelled multigraph G having as nodes unary predicates or constants from $\mathcal{O}$ and s.t. each edge is labelled with a binary predicate from $\mathcal{O}$ or type. Each edge e is justified by a fact or rule α_e s.t. $\mathcal{O} \models \alpha_e$ and α_e is of the form given next, where c, d are constants, A, B unary predicates and R a binary predicate:

(i) if e is $c \xrightarrow{R} d$, then α_e is of the form

$R(c, d)$ or $R(c, y) \rightarrow y \approx d$;

(ii) if e is $c \xrightarrow{R} A$, then α_e is a rule of the form

$\top(c) \rightarrow \exists y[R(c, y) \wedge A(y)]$ or $R(c, y) \rightarrow A(y)$;

(iii) if e is $A \xrightarrow{R} c$, then α_e is a rule of either of the form

$A(x) \rightarrow R(x, c)$ or $A(x) \wedge R(x, y) \rightarrow y \approx c$;

(iv) if e is $A \xrightarrow{R} B$, then α_e is a rule of the form

$A(x) \rightarrow \exists y[R(x, y) \wedge B(y)]$ or $A(x) \wedge R(x, y) \rightarrow B(y)$;

(v) if e is $c \xrightarrow{\text{type}} A$, then $\alpha_e = A(c)$.

Moreover, $\text{range}_G(R)$ denotes the set of nodes in G with an incoming R-labelled edge.

The first (resp. second) option for each α_e in (i)–(iv) encodes the existential (resp. universal) R-relation between nodes in e, whereas (v) encodes typing. A graph may not contain all justifiable edges, but rather those that are deemed relevant to the given application.

Example 29. Recall our ontology in Example 1. A facet graph may contain nodes for :*bc* (Bill Clinton) and :*cc* (Chelsea Clinton), as well as for predicates such as USpres and Univ. Example edges are: (i) a child-edge linking Bill Clinton to Chelsea Clinton, which is justified by the fact child(:*bc*, :*cc*); (ii) a citiz-edge from Person to Country justified by Rule (4); and (iii) a grad-edge from :*cc* to Univ since Chelsea Clinton graduated from Stanford and therefore the ontology entails the sentence $\text{Person}(:cc) \rightarrow \exists y(\text{grad}(:cc, y) \wedge \text{Univ}(y))$.

It follows from the following proposition that facet graph computation can be efficiently implemented. In practice, the graph can be precomputed offline when first loading data and ontology. It can then be stored in RDF and accessed using SPARQL queries during search.

Proposition 30. *Checking whether a directed labelled multigraph is a facet graph for $\mathcal{O}$ is feasible in polynomial time if $\mathcal{O}$ is in any of the OWL 2 profiles.*

Proof. It suffices to show that checking whether an edge in the graph is justified is feasible in polynomial time. We show that checking entailment for each different type of rule or fact α is feasible in polynomial time for all profiles.

- $\alpha_e = R(c, d)$ and $\alpha_e = A(c)$. As already discussed, fact entailment is tractable for all profiles.
- α_e is a Datalog rule $\gamma_1 \dots \gamma_n \rightarrow \eta$. Consider a substitution $\sigma = \{x \mapsto e, y \mapsto f\}$ with e and f fresh constants not occurring in $\mathcal{O}$. Then, $\mathcal{O} \models \alpha_e$ iff $\mathcal{O} \cup \{\sigma(\gamma_i)\}_{i=1}^{n} \models \sigma(\eta)$. Tractability of checking $\mathcal{O} \models \alpha_e$ then follows immediately from tractability of fact entailment in the profiles.
- $\alpha_e = A(x) \rightarrow \exists y[R(x, y) \wedge B(y)]$. Tractability of checking $\mathcal{O} \models \alpha_e$ follows from tractability of subsumption checking for EL and QL. In the case of RL we have that $\mathcal{O} \models \alpha_e$ iff $\mathcal{O} \cup \{A(e)\} \models \exists y[R(e, y) \wedge B(y)]$, in which case tractability follows from tractability of tree-shaped CQ evaluation for RL.
- $\alpha_e = \top(c) \rightarrow \exists y[R(c, y) \wedge A(y)]$. We have that $\mathcal{O} \models \alpha_e$ iff $\mathcal{O} \cup \{\top(c)\} \models \exists y[R(c, y) \wedge A(y)]$. The argument is then the same as in the previous case for RL. If we consider EL and QL, we have that $\mathcal{O} \cup \{\top(c)\} \models \exists y[R(c, y) \wedge A(y)]$ iff c is an instance of the concept $\exists R.A$ w.r.t. $\mathcal{O}$, a tractable problem for both EL and QL. □

To realise the idea of ontology-guided faceted navigation, we require that interfaces *conform* to the facet graph, in the sense that the presence of every facet and value in the interface is supported by a graph edge. In this way, we ensure that interfaces mimic the structure of (and implicit information in) the ontology and the interface does not contain irrelevant (combinations of) facets. Since a given facet or value can occur in many different places in an interface, we need a mechanism for unambiguously referring to each element in the interface. To this end, we introduce an alternative representation of interfaces in the form of a tree. This representation will also be instrumental to our notions of update in Section 5.3.

Definition 31. The node-labelled tree $\text{tree}(I) = (N, E, \lambda)$ of a simple EFI I is recursively defined as follows.

(i) If I is an EBFI, then $N = \{\varepsilon\}$, $E = \emptyset$, and $\lambda(\varepsilon) = I$.

(ii) If $I = (I_0 \wedge I_1)$ where $\text{tree}(I_i) = (N_i, E_i, \lambda_i)$, then

$N = \{\varepsilon\} \cup \{0w \mid w \in N_0\} \cup \{1w \mid w \in N_1\}$,

$E = \{(\varepsilon, 0), (\varepsilon, 1)\} \cup \{(iu_1, iu_2) \mid (u_1, u_2) \in E_i\}$.

Furthermore, $\lambda(w) = \varepsilon$ if $w = \varepsilon$, and $\lambda(w) = \lambda_i(u)$ if w of the form iu with $i \in \{0, 1\}$.

(iii) If $I = (I_0/I_1)$, where $\text{tree}(I_1) = (N_1, E_1, \lambda_1)$, then

$N = \{\varepsilon\} \cup \{0w \mid w \in N_1\}$,

$E = \{(\varepsilon, 0)\} \cup \{(0u_1, 0u_2) \mid (u_1, u_2) \in E_1\}$.

Furthermore, $\lambda(\varepsilon) = I_0$, and for each $w \in N \setminus \{\varepsilon\}$ it holds that $\lambda(w) = \lambda_1(u)$ where $w = 0u$.

A *position* in I is a pair (w, v) where w is a node in $\text{tree}(I)$ with label an EBFI (F, Σ) and $v \in F|_2 \cup \{\text{focus}\}$.

We can now define conformance of interfaces to facet graphs.

Definition 32. Let G be a facet graph for $\mathcal{O}$ and I a simple EFI. Let (w_1, v_1) and (w_2, v_2) be distinct positions in I, where $\lambda(w_i)$ in $\text{tree}(I)$ is (F_i, Σ_i) and $F_i|_1 = X_i$ for $i = 1, 2$. Position (w_2, v_2) is *justified* by (w_1, v_1) in G if w_1 is the least ancestor of w_2 in $\text{tree}(I)$ with $\lambda(w_1) \neq \varepsilon$ and one of the following properties holds: (i) there is an X_2-labelled edge from v_1 to v_2; or (ii) $v_1 = \text{any}$ and there is an X_2-labelled edge from some $u \in \text{range}_G(X_1)$ to v_2; or (iii) $v_2 = \text{any}$ and v_1 has an outgoing X_2-edge; or (iv) $v_1 = v_2 = \text{any}$ and u has an outgoing X_2-edge for some $u \in \text{range}_G(X_1)$.

Interface I *conforms to* G if for each position (w, v) in I, either (i) there is no ancestor w' of w in $\text{tree}(I)$ with $\lambda(w) \neq \varepsilon$; or (ii) there is a position (w', v') in I s.t. $\lambda(w')$ is (F', Σ'), $v' \in \Sigma'$ and (w, v) is justified by (w', v') in G.

Intuitively, (w_2, v_2) is justified by (w_1, v_1) if there is an edge from v_1 to v_2 labelled with the facet predicate X_2 of F_2. This indicates that there is an entailment in $\mathcal{O}$ that justifies the appearance of v_2 given v_1 and X_2. Our definition, however, must also consider that v_1 can be any, which indicates that any value reachable by using the facet predicate X_1 of facet F_1 can be used to justify v_2. Analogously, v_2 can also be any, in which case it is enough to use v_1 to justify any value reachable by using the facet predicate X_2.

Algorithm 3: CreateInterface

INPUT : A facet graph $G = (V, E)$ for $\mathcal{O}$, a set S of nodes in G
OUTPUT : A simple faceted interface

1 $\Upsilon = \{w \mid v \xrightarrow{\text{type}} w \in E \text{ and } v \in S\}$
2 $I = ((\text{type}, \vee\Upsilon), \emptyset)$
3 **for each** $R \in \mathbf{BP}$ **do**
4 $\quad \Gamma, \Upsilon' := \emptyset$
5 $\quad$ **for each** $v \in S$ *and* $v \xrightarrow{R} w \in E$ **do**
6 $\quad\quad$ **if** w *is a constant* **then** $\Gamma := \Gamma \cup \{w\}$
7 $\quad\quad$ **else** $\Upsilon' := \Upsilon' \cup \{w\}$
8 $\quad$ **if** $\Gamma \neq \emptyset$ **then** $I := I \wedge ((R, \vee(\Gamma \cup \{\text{any}\})), \emptyset)$
9 $\quad$ **if** $\Upsilon' \neq \emptyset$ **then** $I := I \wedge ((R, \vee(\Upsilon' \cup \{\text{any}\})), \emptyset)$
10 **return** I

5.2. Interface generation

Algorithm 3 shows how a fresh interface can be generated from a starting set S of nodes in a facet graph G. The algorithm starts by grouping all unary predicates categorising the constants in S in a BFI (Lines 1–2). Then, for each binary predicate R and each $v \in S$, the algorithm collects the nodes w with an incoming R-edge from v and groups them in sets Γ and Υ' depending on whether they are constants or unary predicates (Lines 3–7). All constants in Γ (resp. predicates in Υ') are put together in a BFI with facet predicate R, which is coupled to the interface using $\wedge$-branching (Lines 8–9).

Algorithm 3 can be directly exploited to generate an initial interface from a set of keywords. A faceted search back-end would first compute an initial set D of entities relevant to the keywords (e.g., using a text search engine), and then generate an initial interface by calling Algorithm 3 with input D and a facet graph for $\mathcal{O}$. The resulting interface I has no selected facet values or nested facets, which reflects that I constitutes the starting point to navigation. Furthermore, I is conformant to the input graph G.

Proposition 33. *On input G and S, Algorithm 3 outputs a simple interface that conforms to G.*

Proof. By construction, the output interface I contains only disjunctive facets and does not contain subfacets of the form $(path_1 \vee path_2)$; thus the algorithm outputs a simple interface. Now, note that I is a conjunction of BFIs and hence no position in I has an ancestor w' with $\lambda(w') \neq \varepsilon$. This proves the conformance of I to G and concludes the proof. □

5.3. Interface update

The initial interface where no facet value has been yet selected marks the start of the navigation process. We define the elementary operations on facet values by exploiting the tree representation of interfaces (c.f. Definition 31). We start with the selection operation.

Definition 34. The action Select is applicable to a simple EFI I, a position (w, v) in I, and a facet graph G for $\mathcal{O}$ under the following preconditions: (i) v is not selected in $\lambda(w)$ and (ii) if an ancestor w' of w in $\text{tree}(I)$ is labelled with an EBFI (F', Σ'), then $\Sigma' \neq \emptyset$. The result is the interface computed by Algorithm 4.

Algorithm Select starts by checking whether the value v is focus, in which case it adds v to Σ and removes all other occurrences of focus in I (Lines 1–2). Otherwise, it generates a fresh EFI I_1 from I by adding v to Σ (Line 4), and constructs a new EFI I_2 that collects all the values adjacent to v in G (Line 5). Notice that if $v = \text{any}$, then the value v itself is not considered; instead, v is replaced by the values in G with an incoming $F|_1$-labelled edge.

Algorithm 4: Select

INPUT : I, (w, v), and G as in Def. 34, with $\lambda(w) = (F, \Sigma)$
OUTPUT : A simple EFI

1 **if** $v = \text{focus}$ **then**
2 $\quad I_{out}$:= remove all occurrences of focus in I, and then replace Σ in $\lambda(w)$ with $\Sigma \cup \{\text{focus}\}$
3 **else**
4 $\quad I_1$:= replace Σ in I with $\Sigma \cup \{v\}$
5 $\quad$ **if** $v \in \mathbf{C} \cup \mathbf{UP}$ **then** $I_2 := \text{CreateInterface}(G, \{v\})$
6 $\quad$ **else** $I_2 := \text{CreateInterface}(G, \text{range}_G(F|_1))$
7 $\quad$ **if** w *is a leaf in* $\text{tree}(I_1)$ **then**
8 $\quad\quad I_{out}$:= replace $\lambda(w)$ in I_1 with $(\lambda(w)/I_2)$
9 $\quad$ **else** I_{out} := replace $\lambda(w0)$ in I_1 with $(\lambda(w0) \wedge I_2)$
10 **return** I_{out}

Algorithm 5: Unselect

INPUT : I, (w, v) and G as in Def. 36, with $\lambda(w) = (F, \Sigma)$
OUTPUT : A simple EFI

1 **if** $v = \text{focus}$ **then** I_{out} := replace Σ in I with $\Sigma \setminus \{\text{focus}\}$
2 **else**
3 $\quad S := \{(w', v') \mid (w', v')$ is uniquely justified by (w, v) in G, $\lambda(w') = (F', \Sigma')$ and $v' \in \Sigma'\}$
4 $\quad$ **for each** $(w', v') \in S$ **do** $I := \text{Unselect}(I, (w', v'), G)$
5 $\quad I_{out}$:= replace Σ in I with $\Sigma \setminus \{v\}$
6 λ_{out} := labelling function of $\text{tree}(I_{out})$
7 **for each** *node* w' *in* $\text{tree}(I_{out})$ **do**
8 $\quad (F', \Sigma') := \lambda_{out}(w')$
9 $\quad$ **if** $\lambda_{out}(w'') = (F'', \emptyset)$ *for some ancestor* w'' *of* w' *in* $\text{tree}(I_{out})$ **then** I_{out} := replace Σ' in I_{out} with $\emptyset$
10 **return** I_{out}

Finally, Algorithm Select includes in I_1 the navigation alternatives encoded in I_2 by considering two cases. If w is a leaf in $\text{tree}(I_1)$, then we incorporate I_2 via nesting by replacing $\lambda(w)$ in I_1 with $(\lambda(w)/I_2)$ (Line 7); otherwise, w has a nested child $w0$ in $\text{tree}(I_1)$, in which case the navigation alternatives encoded in I_2 are included in $w0$ by replacing $\lambda(w0)$ in I_1 with $(\lambda(w0) \wedge I_2)$.

Proposition 35. *Assume that I, (w, v) and G are as in Definition 34. If I conforms to G, then* Select$(I, (w, v), G)$ *is a simple EFI that also conforms to G.*

Proof. Clearly, the output interface I_{out} is simple since (i) the input interface I is simple, (ii) the modifications in Lines 1–6 do not affect the simplicity, (iii) the only new subinterface I_2 which is added in Line 8 or 9 consists of disjunctive facets, and (iv) no subinterface of the form $(path_1 \vee path_2)$ is added.

Now we turn to the conformance to G. Since the input interface I conforms to G, we need to check the conformance conditions only for those positions in I_{out} that correspond to I_2. Let (w_2, v_2) be a such position; then it is easy to see that (w, v) justifies (w_2, v_2). Indeed, if $v \neq \text{any}$, then (i) the least ancestor of w_2 is w, (ii) if $v_2 \neq \text{any}$, then it occurs in I_2 only if there is a $F|_1$-labelled edge from v to v_2, where F is a facet in $\lambda(w)$ (see Lines 5–7 in Algorithm 3), and (iii) if $v_2 = \text{any}$, then it occurs in I_2 only if v has an outgoing $F|_1$-labelled edge (see Lines 8–9 in Algorithm 3). The case $v = \text{any}$ is analogous. □

We next define what it means to unselect a facet value. Intuitively, when unselecting v in a given position of an interface all values that were justified by v (and only by v) should also be unselected. In particular, we say that (w_2, v_2) is *uniquely justified* by (w_1, v_1) in G if (w_2, v_2) is justified by (w_1, v_1) in G and (w_2, v_2) is not justified in G by any pair other than (w_1, v_1).

Definition 36. The action Unselect is applicable to a simple EFI I, a position (w, v) in I and a facet graph G for an ontology $\mathcal{O}$, if $v \in \Sigma$

with (F, Σ) the label of w in tree(I). The result is the interface computed by Algorithm 5.

Algorithm UNSELECT considers two cases depending on what kind of value v is unselected. If v is focus, then the value is simply unselected (Line 1). Otherwise, not only Σ must be replaced in I with $\Sigma \setminus \{v\}$, but also all the positions in I that are uniquely justified by (w, v) have to be unselected (Lines 2–5). Unselecting a value propagates recursively along the tree of I since positions deeper down the tree could ultimately be affected. Finally, the algorithm makes sure that no selected value remains disconnected to the rest (Lines 7–9).

Proposition 37. *Assume that I, (w, v) and G are as in Definition 36. If I conforms to G, then* UNSELECT$(I, (w, v), G)$ *is a simple EFI that also conforms to G.*

Proof. Note that the algorithm modifies only sets of selected values Σ in some EBFIs occurring in the input interface I, which immediately yields that I_{out} inherits simplicity and the conformance to G from I. □

5.4. Minimising interfaces

An important issue in the design of faceted interfaces is to avoid the overload of users with redundant facets or facet values. Intuitively, an (unselected) facet value v is redundant if selecting v either leads to a 'dead end' (i.e., an empty set of answers) or it does not have an effect on query answers. Then, a faceted interface is minimal if none of its component BFIs contains redundant values.

Definition 38. Let I be a simple EFI and G a facet graph for $\mathcal{O}$. Then I is minimal w.r.t. G if for each position (w, v) in I s.t. SELECT is applicable to I, (w, v) and G, the following holds: (i) $Q[\text{SELECT}(I, (w, v), G)]$ has a non-empty answer set w.r.t. $\mathcal{O}$; and (ii) the answers to $Q[\text{SELECT}(I, (w, v), G)]$ w.r.t. $\mathcal{O}$ are different from the answers to $Q[I]$ w.r.t. $\mathcal{O}$.

Example 39. The transition from interface I_0 to I_1 in Example 26 involves a minimisation step. The BFI in I_0 involving F_5 is pruned since selecting a value will either not affect the search results (if any or :*us* is selected) or yield an empty set of answers (if :*uk* is selected).

To avoid overwhelming users with irrelevant information, systems can minimise the output of Algorithm 4 before showing it to the user.

6. SemFacet: a faceted search system

We next describe our faceted search system SemFacet, which is implemented in Java and available for download under an academic licence. The system can be obtained from our project website [48], where we also provide a collection of test data and detailed installation and configuration instructions.[5] In this section, we also report on a proof of concept performance evaluation as well as on our practical experience with Yago.

6.1. System description

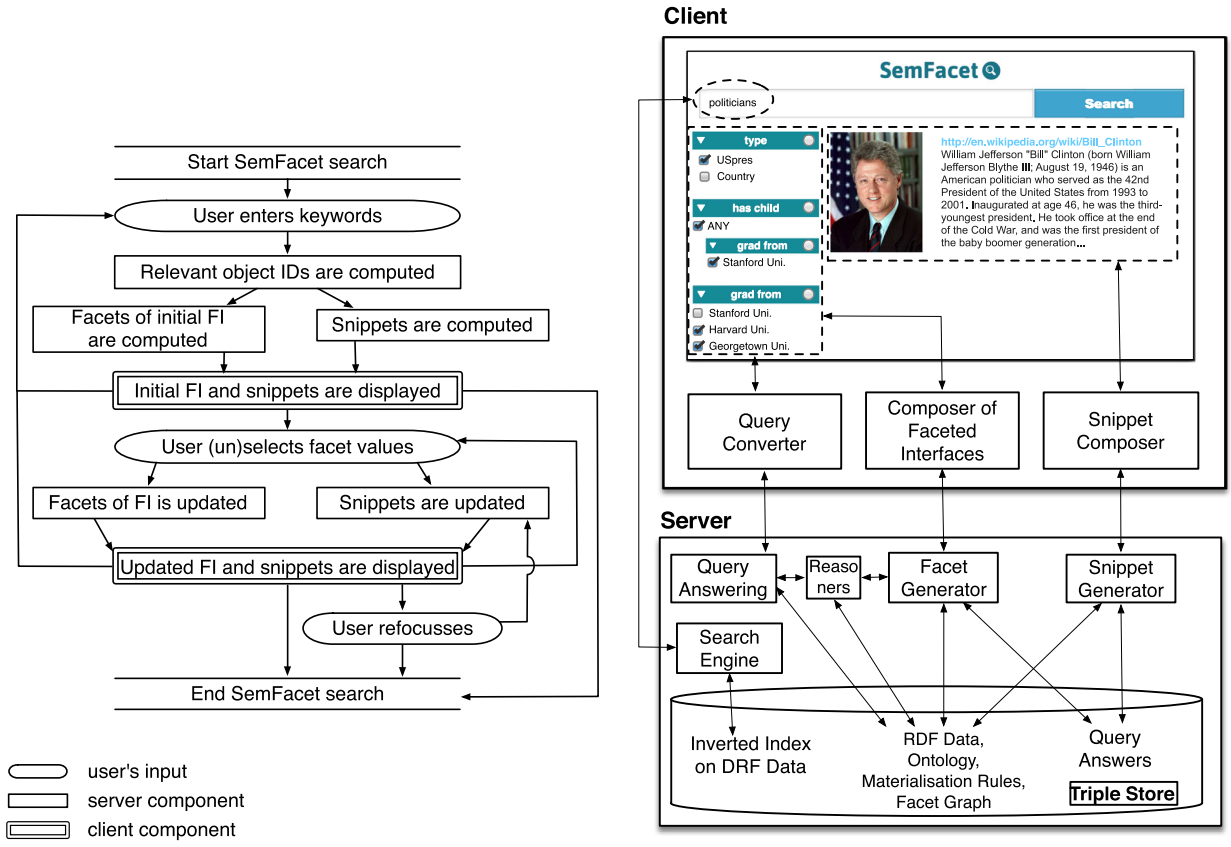
System overview. SemFacet's workflow is summarised in Fig. 2(a), where the steps relevant to users' activity are depicted as ovals, and those relevant to system's activity are represented as boxes (double-lined for front-end tasks and single-lined for back-end tasks). Users initiate the search by entering a set of keywords, which are then matched to textual information associated to URIs in the data (such as labels and descriptions) resulting in an initial set of *relevant URIs*.[6] SemFacet then computes the initial interface (with no value selections) based on these relevant URIs, which constitutes the starting point for faceted navigation. We now provide further details on how the main tasks performed by SemFacet are realised in the system.

- *Matching of keywords.* SemFacet exploits the values of annotation properties to determine whether a URI is relevant to a set of keywords. Roughly speaking, a URI u is relevant to a keyword k w.r.t. an annotation property R if the input data contains a triple of the form (u, R, w), where w is a string containing k. Furthermore, u is relevant to a set of keywords if at least one of them occurs in w. To implement keyword search, SemFacet constructs an inverted index on the strings occurring in the values of these annotation properties. Alternatively, the system can be configured to rely on existing search engines such as Lucene [50] and delegate keyword search to them.
- *Interface generation and update.* SemFacet relies on a facet graph G of the input RDF data and ontology to generate and update faceted interfaces. The part of the graph corresponding to entailed facts (i.e, edges of Type (i) and (v) in Definition 28) is materialised offline at loading time. Edges of Types (ii)–(iv) in Definition 28 are computed in the online phase by querying the materialised graph. The initial interface is generated according to Algorithm 3 by isolating in G the nodes corresponding to the URIs returned by the keywords, the edges outgoing from them, and the nodes reached by these edges. Faceted interfaces are updated in response to user actions using Algorithms 4 and 5; moreover, SemFacet relies on the strategies described in Section 5.4 for interface minimisation. Specifically, our system executes each possible expansion of an EFI in the background by calling the reasoner, and prunes all facet values that either do not change query answers, or make them empty. Finally, the current version of the system can be customised so that facet values are hierarchically arranged according to a user-specified predicate, which greatly facilitates navigation in the presence of a large number of values per facet.
- *Query generation and execution.* SemFacet compiles faceted queries obtained from user selections in an interface into SPARQL queries, which are then evaluated using a reasoner. Our system currently bundles several reasoning engines with different capabilities, and users can select the reasoner that is deemed more appropriate for their application at hand. Answers to SPARQL queries are typically returned by reasoners in the form of a URI. This may not be very informative for end users; hence, SemFacet also displays the annotations associated to the answer URIs and displays them in the form of a snippet.

System architecture. Our system is based on a modular architecture, which is depicted in Fig. 2(b). On the client side, SemFacet implements a GUI developed using HTML 5 consisting of three main parts: a free text search box for keywords, a hierarchically organised faceted interface, and a scrollable panel containing snippet-shaped answers. User keywords are sent by the client to the server where they are processed by the search engine. For efficiency reasons, we implemented our own simple engine based on an inverted index, and also allowed for the possibility of delegating keyword search to Lucene [50].

[5] SemFacet is also available on GitHub [49].

[6] If the given set of keywords is empty, the system considers all URIs in the data as relevant.



(a) Workflow diagram of SemFacet (FI stands for Faceted Interface).

(b) Architecture of SemFacet.

Fig. 2. Workflow diagram and architecture.

User selections in the faceted interface are compiled into a SPARQL query using the *query converter* and then sent to the back-end reasoner for evaluation. The *snippet and interface composers* receive information about facets and answers that should be displayed to the user and update the currently displayed interface and query answers. The system updates the faceted interface incrementally: only the parts of the interface that are affected by users' actions are updated, which allows for a significantly faster response time. On the server side, the system relies on an *in-memory triple store* to store the inverted index, input data and ontology, facet graph, and query answers. The current implementation bundles JRDFox [51,52],[7] Sesame [53,54],[8] Stardog [55,56],[9] PAGOdA [57–59],[10] and HermiT [60].[11] Any other in-memory triple store providing similar functionality can be seamlessly integrated with SemFacet. Please note that SemFacet requires that all data be stored in main memory, which may limit the applicability of the system. We are currently working on scalable solutions that would involve access to secondary storage; a first step in this direction would be to store on disk the inverted index used for keyword matching as well as the annotations relevant to snippet generation.

The *facet generator* is the back-end component responsible for constructing the interface in response to user actions, while the *query answering component* of the back-end executes the SPARQL query obtained from the query converter using the reasoning engine selected by the user.

Configuring the system. SemFacet offers a range of options for system administrators to deploy and configure the system (see Fig. 3 for a screenshot of the system's configuration manager). These include (i) the reasoning engine of choice (JRDFox, PAGOdA, Sesame, Stardog, or HermiT); (ii) the annotation properties relevant for keyword search and displaying of query answers; and (iii) the facet that is first displayed to the user. By default, values within a facet are interpreted disjunctively; however, SemFacet provides advanced configuration capabilities for specifying which facets must be interpreted conjunctively. Additionally, the hierarchical display of facet values can also be configured by specifying the property used to construct the hierarchy (typically *rdfs:subClassOf* or a property capturing a partonomy relation).

6.2. Performance evaluation

We have evaluated the performance of interface generation and update in SemFacet using different triple stores on the system's back-end. The main goal of our experiments was to assess the practical feasibility of our approach when implemented on top of widely-used triple stores with reasoning capabilities, rather than to benchmark the triple stores themselves.

Performance metrics. Interface generation as described in Algorithm 3 requires computing all triples (v, w, u) in the facet graph G for

[7] JRDFox is an in-memory RDF triple store that supports shared memory parallel Datalog reasoning. It is written in C++ and comes with a Java wrapper allowing for a seamless integration with Java-based applications.

[8] Sesame is a widely-used Java framework for processing RDF data. It offers an easy-to-use API that can be connected to all leading RDF storage solutions.

[9] Stardog is a Java-based triple store providing reasoning support for all OWL 2 profiles as well as a SPARQL implementation.

[10] PAGOdA is a query answering system that exploits a hybrid approach to answer CQs over OWL 2 ontologies and combines a Datalog reasoner with a fully-fledged OWL 2 reasoner in order to provide scalable 'pay as you go' performance.

[11] HermiT is the first publicly-available OWL reasoner based on a novel 'hypertableau' calculus which provides much more efficient reasoning than any previously-known algorithm.

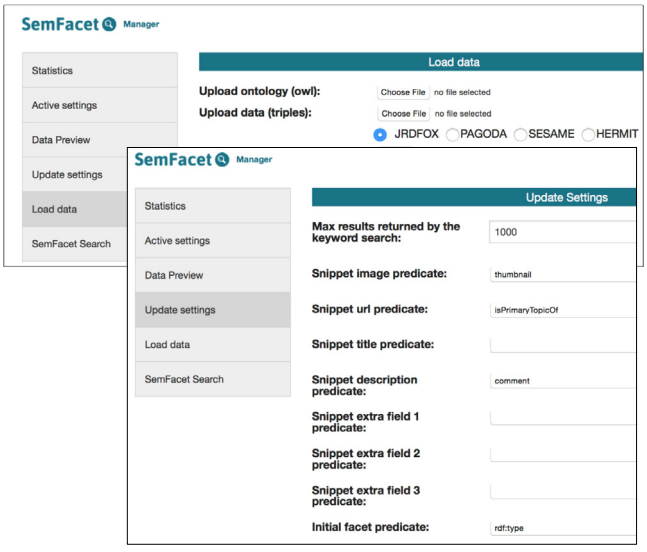


Fig. 3. Configuration manager of SemFacet.

Algorithm 6: CreateInterfaceNaive

INPUT : G: facet graph; S: set of nodes in G
OUTPUT: A simple faceted interface

1 I := Empty interface
2 **for each** $v \in S$ **do**
3 | Pairs_v := Select ?y,?z From G Where (v, ?y, ?z)
4 | **for each** $t \in \text{Pairs}_v$ **do** I := ComposeInterface(t, I)

each v in the input nodes S, and then iterating over the results to compose the interface. Thus, performance of our system critically depends on the following parameters of the underlying triple store, which can be estimated empirically by benchmarking the triple store over the dataset of interest:

- $t[\textit{run query}]$: time to execute an atomic query; and
- $t[\textit{look up}]$: time to iterate over query results.

We implemented Algorithm 3 using two approaches: naive and lazy. A naive approach is described in Algorithm 6: for each $v \in S$, it retrieves relevant pairs (w, u) by a single SPARQL query to the store on the server side, and it uses a routine ComposeInterface to construct the faceted interface on the client side. For improved efficiency, our system implements a variation of Algorithm 6 where facets are computed lazily: facet predicates are computed first, and values are computed on demand when users click on a facet. For this, we modify the query in Line 3 such that ?y is the only answer variable.

To estimate the cost of interface generation (t_{CI}), we estimate the cost of Algorithm 6 and its lazy version. We are interested only in the cost of the server computations and thus assume constant time for the call to ComposeInterface. The cost can then be estimated as follows:

$$t_{CI} = (|S| \times t[\textit{run query}]) + (\#[\textit{answers}] \times t[\textit{look up}]). \quad (5)$$

In this expression, $\#[\textit{answers}]$ is the union of all sets Pairs_v for each $v \in S$. In the worst-case, $\#[\textit{answers}]$ is $|G|$, whereas in the best-case it corresponds to $|S|$. Then, $\#[\textit{answers}]$ is estimated as follows, where the number of facet predicates corresponds to the number of different edge labels in G, and the number of facet values to the number of nodes:

$$\#[\textit{answers}]_{\textit{naive}} = O(\#[\textit{facet predicates}]) \times O(\#[\textit{facet values}]),$$
$$\#[\textit{answers}]_{\textit{lazy}} = O(\#[\textit{facet predicates}]).$$

The cost t_{CI} in Eq. (5) can also be used to estimate the cost of interface updates. Algorithm 4 for selecting a facet value can be seen as a variant of Algorithm 6 with S the set of values relevant to the selection. In the case of unselecting a value, the worst-case cost for Algorithm 5 is estimated as $k \times t_{CI}$, with k the number of selected values in the interface. Indeed, k measures the worst-case number of recursive calls to Unselect, whereas t_{CI} estimates the cost of a single recursive call.

Experimental setup. To estimate the parameters $t[\textit{run query}]$ and $t[\textit{look up}]$, thus also estimating the cost t_{CI} of interface generation, we have conducted experiments over a fragment of DBpedia enriched with RL rules and we have used JRDFox, Stardog, and Sesame as underpinning triple stores. All experiments were conducted on a MacBook Pro laptop with OS X 10.8.5, 2.4 GHz Intel Core i5 processor, and 8 GB 1333 MHz DDR3 memory. Since the triple stores bundled in SemFacet operate in main memory, and we wanted to test our algorithms on stock hardware, we considered a fragment that covers 20% of DBpedia (3.5 million triples) and which can be loaded with 8 GB of RAM. Each experiment was executed 100 times; we measured average and median running time for each experiment. Since results never differ in more than 5% for a single experiment, we report only average times. Please note that our experiments were conducted locally on a single machine and hence do not take into account important factors in client–server architectures such as number of clients, or network usage and bandwidth. In this sense, our experimental results reflect a best possible scenario in terms of performance.

Evaluation results. Results are summarised in Fig. 4. Fig. 4(a) estimates $\#[\textit{answers}] \times t[\textit{look up}]$ by measuring time required to iterate over an answer set of a given size. In turn, Fig. 4(b) estimates $|S| \times t[\textit{run query}]$ by computing the times required for the triple store to answer a given number of atomic queries. We can make the following observations:

- The time needed to iterate over query results is small in comparison to query execution times. For example, to run 10,000 queries, JRDFox requires 0.498 s, whereas to iterate over 10,000 answers it requires 0.002 s. This should be taken into account when optimising interface generation.
- In some triple stores (i.e., Stardog and Sesame), iteration and query answering times do not grow linearly, and they have to be determined empirically. In contrast, JRDFox shows linear behaviour.

We first discuss query execution times. To generate the initial interface, the size of S is determined by the number of relevant results returned by the search engine from keywords. If the ranking algorithm of the search engine produces high quality results, one can establish a cap on S and the system allows for this cap to be set via its Configuration Manager (see the screenshot in Fig. 3 where the cap is set to 1000). As shown in Fig. 4(b), obtaining a reasonable cap is important since query execution is expensive. For example with a cap of 1000 results in S, JRDFox would execute the queries necessary for interface generation almost instantaneously.

Concerning iteration times over query results, JRDFox could perform this task in 0.2 s for 1 million results and 2 s for 10 million. We could not conduct experiments with 10 million answers over Stardog and Sesame since loading the data in our machine consumed all RAM and system behaviour became unstable. The facet graph for the whole of DBpedia contains 24 million facet values and 1843 facet predicates [21]. JRDFox would require 5 s in the worst-case to iterate through that many values using the exhaustive algorithm. When computing interfaces lazily, all triple stores would complete the required iteration over facet predicates instantaneously.

#(answers)	JRDFox	Stardog	Sesame
100	0.000	0.010	0.011
1,000	0.000	0.064	0.060
10,000	0.002	0.521	0.294
100,000	0.021	2.934	0.566
1,000,000	0.206	4.475	2.513
10,000,000	2.056	n/a	n/a

(a) Average runtime in seconds for lookup in a set of query answers.

#(queries)	JRDFox	Stardog	Sesame
1	0.000	0.007	0.012
10	0.000	0.188	0.233
100	0.004	2.414	0.630
1,000	0.059	5.666	3.683
10,000	0.498	15.025	26.126
100,000	4.799	n/a	n/a

(b) Average runtime in seconds for processing a set of queries.

Fig. 4. Experimental results for JRDFox, Stardog, and Sesame.

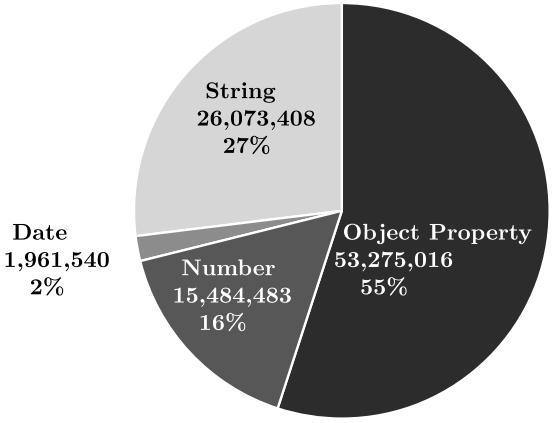


Fig. 5. Distribution of triples in our Yago slice with 96,794,447 triples, before computation of facet graph.

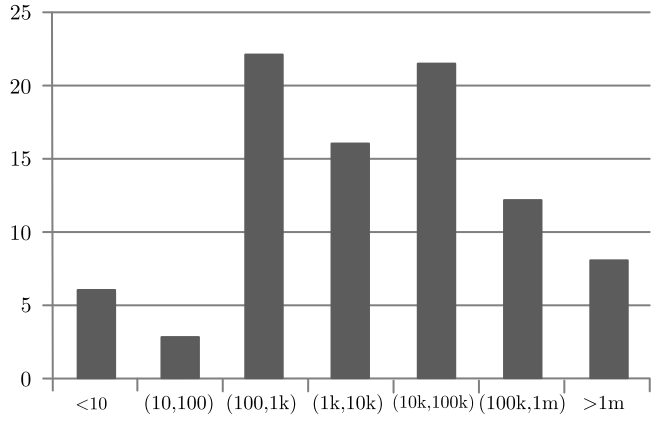


Fig. 6. Distribution of popularity across 89 facet predicates in FYago. On the horizontal axis: popularity values divided in 7 groups; on the vertical axis: number of facets that fell into each group.

6.3. Faceted search over Yago

We have investigated faceted navigation over Yago as a use case. Since the current version of SemFacet relies on main memory triple stores, we did experiments with slices of Yago that could fit in the main memory of our machine. We used the *Taxonomy* slice, which consists of domain and range restrictions as well as subclass relations, and the *Core* slice, which contains instances of object and annotation properties. The axioms from *Taxonomy* constitute the ontology that we used for experiments. We refer to this ontology together with the data slice we used as *FYago*. To generate snippets, we also included DBpedia abstracts, thumbnails, and links to Wikipedia articles.

Statistics relevant to faceted search. FYago contains 97 million triples involving over 3 million URIs. Fig. 5 shows that 55% of triples relate entities via object properties, 16% relate entities to numbers, 2% relate entities to dates, and 27% relate entities to other kinds of strings. FYago involves 89 predicate URIs: an upper bound to the number of facet predicates. We analysed the following measures for each facet predicate P:

- *popularity*: the number of entities annotated with P, i.e., those to which facets with predicate P are applicable;
- *value load*: the maximum number of facet values that a facet with predicate P can contain; and
- *filtering power*: the average number of answers to expect when a value in a facet with predicate P is selected.

Popularity determines the number of nodes in the facet graph with outgoing N-edges, and hence how often a facet predicate occurs in an interface. Value load determines the number of nodes with an incoming N-edge, and hence the maximum number of values in a facet. Finally, the filtering power is associated to the average number of nodes with outgoing N-edges pointing to the same value, and hence determines the number of answers obtained after selecting a facet value.

Popularity of facet predicates. FYago contains 8 facet predicates with popularity exceeding 1 million entities; thus, a facet involving such predicate will occur in most search sessions. Additionally, 12 facet predicates with popularity between 100,000 and 1 million, which implies that they will occur rather often. The remaining 69 facet predicates have popularity below 100,000, and thus they will occur rarely; for instance, a facet predicate with popularity 1000 is relevant to 1000 entities only, and hence only to 0.025% of all data triples. A detailed distribution of popularity across facets in Fig. 6. Table 4 depicts the top 20 facets together with their popularity rating. Observe that only 3 out of the top 8 facet predicates (*hasLongitude*, *hasLatitude*, and *rdf:type*) are meaningful for faceted navigation. The remaining facet predicates are either annotations used for keyword search and/or displaying query answers, or they involve URIs from the reserved vocabulary other than *rdf:type*. The latter URIs can be used to improve the GUI (e.g., *rdfs:subClassOf* is used to organise values of type-facets into hierarchies). The remaining 12 predicates in Table 4 are highly relevant for faceted navigation over Yago. Finally, 65 out of the 69 least frequent predicates are also meaningful for faceted navigation. Based on these observations, we prepared three inputs for facet graph computation: (i) FYago as is (with no pre-processing); (ii) the subset of FYago involving only meaningful facets; and (iii) the subset of FYago involving only the 15 most popular meaningful facets. The latter one, which contains 68% of FYago, is the most attractive for navigation since any interface will contain at most 15 facet predicates.

Value load for facet predicates. We computed statistics for the facet predicates from Table 4. The three facet predicates with popularity

Table 4
Statistics on top-20 facet predicates: their popularity, value load (with and without classes), and filtering power.

Facet predicate	Facet values	Popularity	Value load	+Classes	Filtering power
hasLongitude	number	4,775,113	2,419,609	+0	1.97
hasLatitude	Number	4,774,930	1,822,094	+0	2.62
rdfs:subClassOf	–	4,654,976	–	–	–
hasGeonamesEntityID	–	4,615,914	–	–	–
rdfs:label	–	4,084,428	–	–	–
prefLabel	–	2,954,875	–	–	–
isPreferredMeaningOf	–	2,943,554	–	–	–
rdf:type	Class	2,886,451	374,204	+2	7.71
hasGender	Object	923,364	2	+7	461,682.00
hasFamilyName	String	838,669	282,537	+0	2.97
hasGivenName	String	827,681	77,804	+0	10.64
wasBornOnDate	Date	796,090	72,457	+0	10.99
isLocatedIn	Object	668,010	59,245	+19,843	11.28
wasCreatedOnDate	Date	638,398	38,209	+0	16.71
diedOnDate	Date	359,532	56,400	+0	6.37
hasNumberOfPeople	Number	223,079	41,762	+0	5.34
hasWebSite	String	191,952	217,283	+0	1.00
wasBornIn	Object	189,092	13,385	+6,928	14.13
isAffiliatedTo	Object	147,003	18,915	+6,569	7.77
hasArea	Number	129,715	29,137	+0	4.45
		Min	2	9	1.00
		Max	2,419,609	2,419,609	461,682.00
		Average	368,203	370,426	30,785.72
		Median	59,245	72,457	7.74

exceeding a million are overloaded with values (e.g., there are millions of different longitudes and latitudes). We do not see this as a limitation from the perspective of usability since these values can be compactly represented using intervals, where the user can perform selection using sliders. In the system's GUI we follow this approach and display numerical values as intervals. Most other facet predicates can potentially involve thousands of values, but only in the worst-case where no keywords are used and users did not use an initial facet such as *rdf:type* to initially prune the search space. We have experimented with a range of relevant keywords and found that on average they prune over 99% of possible search results; consequently, the number of possible values per facet is considerably reduced. Moreover, these estimations on value load are for facets that are *not* nested; the deeper the nesting of a facet in an interface, the fewer values it will have. Thus, when the user starts faceted navigation using keywords with good selectivity and then refines the search with nested facets, the value load of facets in the interface is expected to be of manageable size. In Table 4, we present two statistics for value load. The first one corresponds to the case where only the data slice of FYago is taken into account. The second corresponds to the case where we take into account the facts derived using the ontology axioms as well. Clearly in the latter case the number of values per facet increases, and the number of extra values per facet predicate is presented in the column +*Class*. Observe that there is no significant difference in the value load between both cases. In Table 4 we also provide minimum, maximum, average, and median values for value load for both cases.

Filtering power. Observe that most values have good filtering power; that is, selecting such value would result in a small number of answers. The only exception is *hasGender*, which only has two values associated to it. Also observe that the values of *hasWebSite* uniquely determine an entity (i.e., entities in FYago have at most one website).

7. Related work

In this section we review the literature on semantic faceted search and describe other approaches to query formulation for RDF and OWL ontologies.

7.1. Semantic faceted search

Faceted search in the context of RDF was pioneered by the Ontogator system [61]. Ontogator was further developed in [62,63] and found applications in the cultural heritage domain [64], as well as in the clinical sciences [65]. In the last few years faceted search has become a popular paradigm for querying RDF data, and many systems have been developed. Prominent examples include mSpace [22], /facet [24], Piggy Bank [25], Tabulator [19], gFacet [23], tfacet [66], Humboldt [26], Parallax [27], Nested Faceted Browser [67], Longwell [68], faceted DBpedia [21], Sewelis [30], X-ENS [20], Broccoli [28], among others [69,70].

Research in this area has so far been systems-centric and has predominantly been driven by efficiency, effectiveness, and usability concerns. In particular, the focus has been on problems such as facet *indexing* [21,71], *ranking* of facets and their values [21,71], *value grouping* [21,71], or *visualisation* [23,66]. In contrast to most of existing work, we have investigated the theoretic underpinnings of RDF faceted search and developed a comprehensive logic-based framework which accounts for the graph-based nature of the RDF data model, and formally captures the query languages underlying the aforementioned systems. Furthermore, our framework goes beyond RDF and also describes the impact of ontologies on faceted search.

Although previous research has focused largely on systems, there have also been several attempts of formalisation [10,29,30, 72–74]. Oren et al. [29] provide an algebraic definition of faceted interfaces by means of operators on sets of entities. Wagner et al. [10] define facets procedurally from a given conjunctive query and dataset. Roughly speaking, a facet for a variable corresponds to the outgoing edges of the data nodes where the variable is mapped when evaluating the query. To formalise faceted navigation, they introduce operations on queries that can be used to add or remove constraints, as well as to capture refocusing. Ferré and Hermann [30] define facets where values are either queries or operators, rather than individuals or literal values. Then, value selection amounts to a syntactic query transformation, rather than to a filter on a set of entities.

We next compare these approaches to ours based on the underlying query languages and the available mechanisms for interface generation and update.

Query languages. None of the aforementioned formalisations provides a precise characterisation of their query language in terms of first-order logic. From the description in Oren et al. [29] we gather that their queries correspond to monadic tree-shaped CQs with the root variable as output, and enhanced with a limited form of epistemic negation. Thus, their language is incomparable to ours since we allow for disjunction, while they support a form of negation. The language of [10] corresponds to tree-shaped CQs and hence it is strictly contained in ours. Finally, Ferre et al. [30,72–74] allow queries as facet values, and these queries can be constrained to any fragment of SPARQL.

The query language underpinning the faceted search systems mentioned in the beginning of this section is rather difficult to understand, given that their description is informal. To the best of our knowledge, most systems support some form of conjunctive nesting, branching, and refocusing [27,28]. A few systems also support a limited form of disjunction [20,21].

Finally, we are not aware of any paper on RDF-based faceted search where the computational complexity of query evaluation is studied. The typical assumption in existing work is that queries are compiled into SPARQL [30] or Prolog [24], and executed by means of an off-the-shelf query evaluation engine.

Interface generation and update. A common approach in existing systems, including [10,29], is to generate and update interfaces from RDF datasets under the assumption that URIs in predicate position correspond to facet predicates while those in object position are facet values. Facets are typically arranged as trees [10, 11,24,29,68] or more complex graphs [23]. Such trees or graphs are, however, defined on RDF data only, and they are also dependent on search results, or the specific GUI of the system. In contrast, our notions of interface and facet graph account for both data and ontologies, and they are independent from search results as well as from the system's GUI. Thus, we see our approach as a generalisation of existing work. Finally, to the best of our knowledge, the complexity of interface generation and update has not been studied in the literature.

7.2. Other query formulation approaches

In recent years, query formulation has been extensively studied by the Semantic Web community. Most of the research has focused on *Visual Query Systems* (VQS), while *natural language* interfaces have also attracted a considerable attention.

Visual query systems. VQS [75] rely on a visual representation paradigm for constructing and modifying queries. Many VQS provide a set of graphical primitives (e.g., boxes, circles, arrows) for query elements (e.g., variables, relations), and a mechanism for combining such primitives into queries. Thus, in VQS, the user is involved in the explicit construction of a query. In contrast, in faceted search, the main focus is on exploration of the underlying data and ontology, rather than on the deliberate construction of a query. Prominent examples of VQS are NITELIGHT [12], SEWASIE [76], iSPARQL [13], OntoVQL [77], Wonder [78], OptiqueVQS [79,80], LUPOSDATE-VEdit [81], and QueryVOWL [14].

Natural language. These systems offer a different approach to query formulation and can be divided in two groups: Question Answering and Controlled Natural Language. The former systems allow users to pose a free text question (or just a set of keywords) and then interpret the input as a formal query. Such systems include FALCON [15], AquaLog [82], AutoSPARQL [83], QuestIO [84], Siemens' query system [85], and SPARK [16]. Systems in the second group, such as Quelo [17], allow for natural language expressions to be used at each step during query construction. Regarding comparison with faceted search, recall that free text in faceted search is only used to initiate the search (indeed, many papers do not discuss text search); in contrast, in natural language systems the text determines the query.

8. Conclusion and future work

In this paper, we have proposed a rigorous theoretical framework for faceted search in the context of RDF-based knowledge graphs enhanced with OWL 2 ontologies. Our framework has allowed us to identify fragments of SPARQL that can be naturally captured using faceted search as a query paradigm, and for which query answering is tractable. Additionally, we have studied the problem of updating faceted interfaces, which is critical for guiding users in the formulation of meaningful queries during exploratory search, and implemented our techniques in a fully-fledged faceted search system.

We see many directions for future work, which we briefly summarise next.

- *Keyword search* could be enhanced by explicitly taking into account the structure of the graph data; this would allow us to compute more suitable initial interfaces. A possible approach in this direction would be to also exploit SPARQL queries for keyword matching.
- *Ranking of facet values.* In our work, we have so far abstracted from GUI-specific considerations; as a next step, we are planning to experiment with a number of ranking algorithms for displaying facets and their values.
- *Formalisation of advanced functionality*. A number of advanced features implemented in existing systems, such as hierarchical facets and epistemic negation, are not currently taken into account in our formal framework. We are planning to extend our results in Sections 3–5 to also capture such features.
- *Query optimisation* is a key challenge; faceted navigation is an interactive process, where instant system response is often required.
- *Expressible queries.* Our algorithms are generic, and they have been designed to query arbitrary RDF-based knowledge graphs. When it comes to specific applications, however, our algorithms do not guarantee that all queries deemed relevant can be effectively constructed via faceted search. This may be because such queries cannot be captured by tree-shaped positive existential formulas, or because they are not easily 'reachable' using the information available in the knowledge graph. Thus, it would be interesting to investigate richer notions of interface that lead to more expressive query languages, as well as techniques for optimising faceted navigation given a set of application-specific queries that are deemed relevant.

Finally, we are currently working with our collaborators at EDF Energy [86], Siemens [87,88], and Statoil [89–91] in the development of faceted search solutions for their semantics-based data management systems. We expect that our interaction with these industrial partners will also provide us with large repositories of realistic queries that we could subsequently use for evaluation and optimisation purposes.

References

[1] F.M. Suchanek, G. Kasneci, G. Weikum, Yago: a core of semantic knowledge, in: Proc. of WWW, 2007, pp. 697–706.
[2] Freebase: an open, shared database of the world's knowledge, http://www.freebase.com/.
[3] Google's Knowledge Graph, http://www.google.co.uk/insidesearch/features/search/knowledge.html.

[4] Facebook's Graph Search, https://www.facebook.com/graphsearcher.
[5] Microsoft's Satori, http://blogs.bing.com/search/2013/03/21/understand-your-world-with-bing/.
[6] Yahoo's Knowledge Graph, www.technobuffalo.com/2014/04/21/yahoo-testing-its-own-version-of-googles-knowledge-graph/.
[7] W3C: Resource Description Framework (RDF), http://www.w3.org/RDF/.
[8] W3C: OWL 2 Web Ontology Language, http://www.w3.org/TR/owl2-overview/.
[9] S. Harris, A. Seaborne, SPARQL 1.1 Query language, W3C Recommendation (21 March 2013).
[10] A. Wagner, G. Ladwig, T. Tran, Browsing-oriented Semantic Faceted Search, in: Proc. of DEXA, 2011, pp. 303–319.
[11] P. Heim, T. Ertl, J. Ziegler, Facet graphs: Complex semantic querying made easy, in: Proc. of ESWC, 2010, pp. 288–302.
[12] A. Russell, P.R. Smart, NITELIGHT: A graphical editor for SPARQL queries, in: Proc. of ISWC (Posters and Demos), 2008.
[13] iSPARQL QBE, http://dbpedia.org/isparql/.
[14] F. Haag, S. Lohmann, S. Siek, T. Ertl, Visual querying of linked data with QueryVOWL, in: Joint Proceedings of SumPre 2015 and HSWI 2014-15, CEUR-WS, 2015.
[15] S.M. Harabagiu, D.I. Moldovan, M. Pasca, R. Mihalcea, M. Surdeanu, R.C. Bunescu, R. Girju, V. Rus, P. Morarescu, FALCON: boosting knowledge for answer engines, in: Proc. of TREC, 2000.
[16] Q. Zhou, C. Wang, M. Xiong, H. Wang, Y. Yu, SPARK: adapting keyword query to semantic search, in: Proc. of ISWC, 2007, pp. 694–707.
[17] E. Franconi, P. Guagliardo, M. Trevisan, S. Tessaris, Quelo: an ontology-driven query interface, in: Proc. of DL, 2011.
[18] D. Tunkelang, Faceted Search, Synthesis Lectures on Information Concepts, Retrieval, and Services, Morgan & Claypool Publishers, 2009.
[19] T. Berners-Lee, J. Hollenbach, K. Lu, J. Presbrey, E. Prudhommeaux, M.M.C. Schraefel, Tabulator redux: Browsing and writing linked data, in: Proc. of LDOW, 2008.
[20] P. Fafalios, Y. Tzitzikas, X-ENS: Semantic enrichment of web search results at real-time, in: Proc. of SIGIR, 2013, pp. 1089–1090.
[21] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, H. Düwiger, U. Scheel, Faceted Wikipedia search, in: Proc. of BIS, 2010, pp. 1–11.
[22] m.c. schraefel, D.A. Smith, A. Owens, A. Russell, C. Harris, M.L. Wilson, The volving mSpace platform: Leveraging the semantic web on the trail of the memex, in: Proc. of Hypertext, 2005, pp. 174–183.
[23] P. Heim, J. Ziegler, S. Lohmann, gFacet: A browser for the web of data, in: Proc. of IMC-SSW, 2008, pp. 49–58.
[24] M. Hildebrand, J. van Ossenbruggen, L. Hardman, /facet: A browser for heterogeneous semantic web repositories, in: Proc. of ISWC, 2006, pp. 272–285.
[25] D. Huynh, S. Mazzocchi, D.R. Karger, Piggy Bank: Experience the semantic web inside your web browser, J. Web Semant. 5 (1) (2007) 16–27.
[26] G. Kobilarov, I. Dickinson, Humboldt: Exploring linked data, in: Proc. of LDOW, 2008.
[27] D.F. Huynh, D.R. Karger, Parallax and companion: Set-based browsing for the data web, 2013. www.davidhuynh.net.
[28] H. Bast, F. Bäurle, B. Buchhold, E. Haußmann, Easy access to the freebase dataset, in: Proc. of WWW, 2014, pp. 95–98.
[29] E. Oren, R. Delbru, S. Decker, Extending faceted navigation for RDF data, in: Proc. of ISWC, 2006, pp. 559–572.
[30] S. Ferré, A. Hermann, Semantic search: Reconciling expressive querying and exploratory search, in: Proc. of ISWC, 2011, pp. 177–192.
[31] SNOMED CT, http://www.ihtsdo.org/snomed-ct.
[32] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, C. Lutz, OWL 2 web ontology language profiles, W3C Recommendation.
[33] M. Arenas, B. Cuenca Grau, E. Kharlamov, Š Marciuška, D. Zheleznyakov, Faceted search over ontology-enhanced RDF data, in: Proc. of CIKM, 2014, pp. 939–948.
[34] B. Cuenca Grau, E. Kharlamov, D. Zheleznyakov, M. Arenas, Š Marciuška, On faceted search over knowledge bases, in: Proc. of DL, 2014, pp. 153–156.
[35] M. Arenas, B. Cuenca Grau, E. Kharlamov, Š Marciuška, D. Zheleznyakov, Enabling faceted search over OWL 2 with SemFacet, in: Proc. of OWLED, 2014, pp. 121–132.
[36] B. Cuenca Grau, E. Kharlamov, Š Marciuška, D. Zheleznyakov, Y. Zhou, Querying life science ontologies with SemFacet, in: Proc. of SWAT4LS, 2014.
[37] M. Arenas, B.C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, Towards semantic faceted search, in: Proc. of WWW (Companion Volume), 2014, pp. 219–220.
[38] M. Arenas, B. Cuenca Grau, E. Kharlamov, Š Marciuška, D. Zheleznyakov, E. Jiménez-Ruiz, SemFacet: Semantic faceted search over Yago, in: Proc. of WWW (Companion Volume), 2014, pp. 123–126.
[39] W3C: SPARQL 1.1 Entailment Regimes, www.w3.org/TR/sparql11-entailment/.
[40] M. Yannakakis, Algorithms for acyclic database schemes, in: Proc. of VLDB, 1981, pp. 82–94.
[41] E. Dantsin, T. Eiter, G. Gottlob, A. Voronkov, Complexity and expressive power of logic programming, ACM Comput. Surv. 33 (3) (2001) 374–425.
[42] G. Stefanoni, B. Motik, I. Horrocks, Introducing nominals to the combined query answering approaches for EL, in: Proc. of AAAI, 2013, pp. 1177–1183.
[43] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, M. Zakharyaschev, The combined approach to ontology-based data access, in: Proc. of IJCAI, 2011, pp. 2656–2661.
[44] G. Stefanoni, B. Motik, Answering conjunctive queries over EL knowledge bases with transitive and reflexive roles, in: Proc. of AAAI, 2015.
[45] M. Krötzsch, S. Rudolph, P. Hitzler, ELP: Tractable rules for OWL 2, in: Proc. of ISWC, 2008, pp. 649–664.
[46] M. Bienvenu, M. Ortiz, M. Simkus, G. Xiao, Tractable queries for lightweight description logics, in: Proc. of IJCAI, 2013, pp. 768–774.
[47] S. Kikot, R. Kontchakov, M. Zakharyaschev, On (in)tractability of OBDA with OWL 2 QL, in: Proc. of DL, 2011.
[48] SemFacet Project Page, http://www.cs.ox.ac.uk/isg/tools/SemFacet/.
[49] GitHub of SemFacet, https://github.com/semfacet.
[50] Lucene, www.lucene.apache.org/.
[51] B. Motik, Y. Nenov, R. Piro, I. Horrocks, D. Olteanu, Parallel materialisation of datalog programs in centralised, main-memory RDF systems, in: Proc. of AAAI, 2014, pp. 129–137.
[52] RDFox, www.cs.ox.ac.uk/isg/tools/RDFox/.
[53] J. Broekstra, A. Kampman, F.v. Harmelen, Sesame: A generic architecture for storing and querying RDF and RDF schema, in: Proc. of ISWC, 2002, pp. 54–68.
[54] Sesame, http://rdf4j.org.
[55] H. Pérez-Urbina, E. Rodríguez-Díaz, M. Grove, G. Konstantinidis, E. Sirin, Evaluation of query rewriting approaches for OWL 2, in: Proc. of SSWS+HPCSW, 2012.
[56] Stardog, http://stardog.com/.
[57] Y. Zhou, Y. Nenov, B.C. Grau, I. Horrocks, Pay-as-you-go OWL query answering using a triple store, in: Proc. of AAAI, 2014.
[58] PAGOdA, http://www.cs.ox.ac.uk/isg/tools/PAGOdA/.
[59] Y. Zhou, B.C. Grau, Y. Nenov, I. Horrocks, Pagoda: Pay-as-you-go abox reasoning, in: Proceedings of the 28th International Workshop on Description Logics, Athens, Greece, June 7–10, 2015.
[60] B. Glimm, I. Horrocks, B. Motik, G. Stoilos, Z. Wang, HermiT: An OWL 2 reasoner, J. Automat. Reason. 53 (3) (2014) 245–269.
[61] E. Hyvönen, S. Saarela, K. Viljanen, Ontogator: Combining view- and ontology-based search with semantic browsing, in: Proc. of XML Finland, 2003.
[62] O. Suominen, K. Viljanen, E. Hyvönen, User-centric faceted search for semantic portals, in: Proc. of ESWC, 2007, pp. 356–370.
[63] J. Kurki, E. Hyvönen, Collaborative metadata editor integrated with ontology services and faceted portals, in: Proc. of ORES, 2010.
[64] E. Hyvönen, E. Mäkelä, M. Salminen, A. Valo, K. Viljanen, S. Saarela, M. Junnila, S. Kettula, Museumfinland—finnish museums on the semantic web, J. Web Semant. 3 (2–3) (2005) 224–241.
[65] E. Hyvönen, K. Viljanen, O. Suominen, Healthfinland—finnish health information on the semantic web, in: Proc. of ISWC, 2007, pp. 778–791.
[66] S. Brunk, P. Heim, tfacet: Hierarchical faceted exploration of semantic data using well-known interaction concepts, in: Proc. of International Workshop on Data-Centric Interactions on the Web, 2011.
[67] D.F. Huynh, The Nested Faceted Browser, 2013. people.csail.mit.edu/dfhuynh/projects/nfb/.
[68] C. Veres, K. Johansen, A.L. Opdahl, Browsing and visualizing semantically enriched information resources, in: Proc. of CISIS, 2010, pp. 968–973.
[69] P. Haase, D.M. Herzig, M.A. Musen, T. Tran, Semantic Wiki search, in: Proc. of ESWC, 2009, pp. 445–460.
[70] S. Buschbeck, A. Jameson, R. Troncy, H. Khrouf, O. Suominen, A. Spirescu, A demonstrator for parallel faceted browsing, in: Proc. of EKAW, 2012.
[71] H. Bast, B. Buchhold, An index for efficient semantic full-text search, in: Proc. of CIKM, 2013, pp. 369–378.
[72] S. Ferré, A. Hermann, Reconciling faceted search and query languages for the semantic web, Int. J. Metadata Semant. Ontol. 7 (1) (2012) 37–54.
[73] S. Ferré, Expressive and scalable query-based faceted search over SPARQL endpoints, in: Proc. of ISWC, 2014, pp. 438–453.
[74] S. Ferré, SPARKLIS: a SPARQL endpoint explorer for expressive question answering, in: Proc. of ISWC, 2014, pp. 45–48.
[75] T. Catarci, M.F. Costabile, S. Levialdi, C. Batini, Visual query systems for databases: A survey, J. Vis. Lang. Comput. 8 (2) (1997) 215–260.
[76] D. Beneventano, S. Bergamaschi, F. Guerra, M. Vincini, The SEWASIE network of mediator agents for semantic search, J. UCS 13 (12) (2007) 1936–1969.
[77] A. Fadhil, V. Haarslev, OntoVQL: A graphical query language for OWL ontologies, in: Proc. of DL, 2007.
[78] D. Calvanese, M. Keet, W. Nutt, M. Rodriguez-Muro, G. Stefanoni, Web-based graphical querying of databases through an ontology: the wonder system, in: Proc. of SAC, 2010, pp. 1388–1395.
[79] A. Soylu, E. Kharlamov, D. Zheleznyakov, E. Jiménez-Ruiz, M. Giese, I. Horrocks, OptiqueVQS: Visual query formulation for OBDA, in: DL, 2014, pp. 725–728.
[80] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, I. Horrocks, OptiqueVQS: Towards an ontology-based visual query system for big data, in: Proc. of MEDES, 2013, pp. 119–126.
[81] J. Groppe, S. Groppe, A. Schleifer, Visual query system for analyzing social semantic web, in: Proc. of WWW (Companion Volume), 2011, pp. 217–220.

[82] V. Lopez, V.S. Uren, E. Motta, M. Pasin, Aqualog: An ontology-driven question answering system for organizational semantic intranets, J. Web Semant. 5 (2) (2007) 72–105.
[83] J. Lehmann, L. Bühmann, Autosparql: Let users query your knowledge base, in: Proc. of ESWC, 2011, pp. 63–79.
[84] D. Damljanovic, V. Tablan, K. Bontcheva, A text-based query interface to OWL ontologies, in: Proc. of LREC, 2008.
[85] M. Sander, U. Waltinger, M. Roshchin, T. Runkler, Ontology-based translation of natural language queries to SPARQL, in: Proc. of Natural Language Access to Big Data, AAAI 2014 Fall Symposium, 2014.
[86] P. Chaussecourte, B. Glimm, I. Horrocks, B. Motik, L. Pierre, The energy management adviser at EDF, in: ISWC, 2013, pp. 49–64.
[87] E. Kharlamov, N. Solomakhina, Ö.L. Özçep, D. Zheleznyakov, T. Hubauer, S. Lamparter, M. Roshchin, A. Soylu, S. Watson, How semantic technologies can enhance data access at siemens energy, in: ISWC, 2014, pp. 601–619.
[88] E. Kharlamov, S. Brandt, M. Giese, E. Jimenez-Ruiz, S. Lamparter, C. Neuenstadt, Ö.L. Özçep, C. Pinkel, A. Soylu, D. Zheleznyakov, M. Roshchin, S. Watson, I. Horrocks, Semantic access to siemens streaming data: the optique way, in: ISWC (Posters and Demos), 2015.
[89] E. Kharlamov, D. Hovland, E. Jimenez-Ruiz, D. Lanti, H. Lie, C. Pinkel, M. Rezk, M.G. Skjæveland, E. Thorstensen, G. Xiao, D. Zheleznyakov, I. Horrocks, Ontology based access to exploration data at statoil, in: ISWC, 2015.
[90] E. Kharlamov, E. Jimenez-Ruiz, C. Pinkel, M. Rezk, M.G. Skjæveland, A. Soylu, G. Xiao, D. Zheleznyakov, M. Giese, I. Horrocks, A. Waaler, Optique: Ontology-based data access platform, in: ISWC (Posters and Demos), 2015.
[91] E. Kharlamov, M. Giese, E. Jiménez-Ruiz, M.G. Skjæveland, A. Soylu, D. Zheleznyakov, T. Bagosi, M. Console, P. Haase, I. Horrocks, et al. Optique 1.0: Semantic access to big data: The case of Norwegian Petroleum Directorate's FactPages, in: ISWC (Posters and Demos), 2013, pp. 65–68.