Faceted Search over Ontology-Enhanced RDF Data

Marcelo Arenas[†]
PUC Chile

Bernardo Cuenca Grau[‡]
University of Oxford

Evgeny Kharlamov[‡]
University of Oxford

Sarunas Marciuska[‡]
University of Oxford

Dmitriy Zheleznyakov[‡]
University of Oxford

ABSTRACT

An increasing number of applications rely on RDF, OWL 2, and SPARQL for storing and querying data. SPARQL, however, is not targeted towards end-users, and suitable query interfaces are needed. Faceted search is a prominent approach for end-user data access, and several RDF-based faceted search systems have been developed. There is, however, a lack of rigorous theoretical underpinning for faceted search in the context of RDF and OWL 2. In this paper, we provide such solid foundations. We formalise faceted interfaces for this context, identify a fragment of first-order logic capturing the underlying queries, and study the complexity of answering such queries for RDF and OWL 2 profiles. We then study interface generation and update, and devise efficiently implementable algorithms. Finally, we have implemented and tested our faceted search algorithms for scalability, with encouraging results.

Categories and Subject Descriptors

H.4.m [**Information Systems Applications**]: Miscellaneous; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods

Keywords

Faceted search; Ontology; OWL 2; RDF; SPARQL; Algorithms

1. INTRODUCTION

The Resource Description Framework (RDF) is the W3C recommendation graph data model for representing information about Web resources, and SPARQL is the standard language for querying RDF. In the last ten years, we have witnessed a constant growth in the amount of available RDF data, and an increasing number of applications are relying on RDF and SPARQL for storing, publishing, and querying data. The functionality of many such applications is enhanced by an OWL 2 ontology: a set of first-order sentences that are used to provide background knowledge about the application

---

domain and enrich query answers with information not explicitly given in the RDF data.

Although the growing popularity of RDF, OWL 2, and SPARQL has been accompanied by the development of better and better query answering engines, writing SPARQL queries is not well-suited for the majority of users. Thus, an important challenge is the development of simple yet powerful query interfaces that capture well-defined fragments of SPARQL.

Faceted search is a prominent approach for querying document[1] collections where users can narrow down the search results by progressively applying filters, called *facets* [1]. A facet typically consists of a property (e.g., 'gender' or 'occupation' when querying documents about people) and a set of possible string values (e.g., 'female' or 'research'), and documents in the collection are annotated with property-value pairs. During faceted search users iteratively select facet values and the documents annotated according to the selection are returned as the search result.

Several authors have proposed faceted search for querying document collections annotated with RDF, and a number of systems have been developed, e.g. [2–10]. The theoretical underpinnings of faceted search in the context of semantic technologies, however, have received less attention [11–13]. In particular, the following key questions have not been satisfactorily addressed (see 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 expressive power and complexity of query languages in the context of faceted search. 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. Furthermore, 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.

Furthermore, 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 can be used to enrich query answers with implicit information, thus enhancing the search for relevant documents. Moreover, they provide schema-level structure, which can be exploited to improve faceted interfaces. Finally, RDF-based faceted search systems are data-centric, and hence cannot be exploited to browse large ontologies or to formulate meaningful queries at the schema level. Our second aim is to address this limitation and provide a framework for faceted search that is also applicable to the wider setting of OWL 2.

---

[1]We use 'document' to refer to any resource referenced by a URI.

$(1)\ A(x) \wedge R(x, y_1) \wedge B(y_1) \wedge R(x, y_2) \wedge B(y_2) \rightarrow y_1 \approx y_2,$

$(2)\ R(x, y) \rightarrow S(x, y),\qquad (3)\ A(x) \rightarrow \exists y.[R(x, y) \wedge B(y)],$

$(4)\ A(x) \rightarrow x \approx a,\qquad (5)\ R(x, y) \wedge S(y, z) \rightarrow T(x, z),$

$(6)\ A(x) \rightarrow B(x),\qquad (7)\ A(x) \wedge B(x) \rightarrow C(x),$

$(8)\ R(x, y) \rightarrow A(x),\qquad (9)\ A(x) \wedge R(x, y) \rightarrow B(y),$

$(10)\ A(x) \rightarrow R(x, a),\qquad (11)\ R(x, a) \rightarrow B(x),$

$(12)\ R(x, y) \rightarrow A(y),\qquad (13)\ R(x, y) \rightarrow S(y, x),$

$(14)\ R(x, y) \wedge B(y) \rightarrow A(x)$

Table 1: Rules corresponding to OWL 2 profiles

In Section 3 we formalise faceted interfaces that are tailored towards RDF and OWL 2 and which capture the key functionality implemented in existing faceted search systems. Our interfaces capture both the combination of facets displayed during search, and the facet values selected by users. In this way, an interface encodes both a query, whose answers constitute the current search results, and the facet values available for further selection. Analogously to existing work on RDF-based faceted search and in contrast to traditional faceted search, our notion of interface allows users to 'navigate' across interconnected collections of documents and establish filters to each of them. 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.

In Section 4 we study the expressivity and complexity of *faceted queries*: queries encoded by faceted interfaces. To this end, we identify a fragment of first-order logic that is sufficient to capture such queries, and study the complexity of query answering in the presence of OWL 2 ontologies. Since OWL 2 reasoning can be computationally expensive and hence significantly affect systems' performance and robustness, we focus on ontologies in the OWL 2 profiles [14]: language fragments with favorable computational properties. For each of these profiles we establish tight complexity bounds and propose practical query answering algorithms.

In Section 5 we study interface generation and update. Existing techniques for RDF are based on exploration of the underlying RDF graph. In this way, by generating facets according to the RDF graph, systems can guide users in the formulation of 'meaningful' queries. We lift this approach by proposing a graph-based representation of OWL 2 ontologies and their logical entailments for the purpose of faceted navigation. 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 a faceted search platform that provides functionality for generating and updating interfaces based on our algorithms in Section 5. Our platform relies on an external triple store with OWL 2 reasoning capabilities, and it is compatible with faceted search GUIs, as well as with text search engines for retrieving documents from keywords. We have tested the scalability of our platform using different triple stores, with encouraging results. As proof of concept, we have integrated our platform in a faceted search system that bundles the triple store JRDFox, the search engine Lucene, and our own faceted search GUI.

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 $\mathbf{C} \cup \mathbf{UP} \cup \mathbf{BP}$. We treat equality $\approx$ as an ordinary predicate in **BP**, and assume that any set of formulae contains the axioms of equality for its signature. We treat $\top$ as a special symbol in **UP**, which is used to

represent a tautology. 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 atom and a *dataset* is a finite set of facts. A *rule* is a sentence of the form $\forall \mathbf{x} \forall \mathbf{z}\, [\varphi(\mathbf{x}, \mathbf{z}) \rightarrow \exists \mathbf{y}\, \psi(\mathbf{x}, \mathbf{y})]$, where **x**, **z**, and **y** are pairwise disjoint tuples of variables, 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 $\mathbf{x} \cup \mathbf{y}$. Universal quantifiers are omitted. The restriction of $\psi(\mathbf{x}, \mathbf{y})$ being non-empty ensures satisfiability of any set of rules and facts, which makes query results meaningful. A rule is *Datalog* if its head has at most one atom and all variables are universally quantified.

OWL 2 defines three *profiles*: weaker languages with favourable computational properties [14]. 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] Thus, we see an *ontology* as a finite set of rules and facts. Table 1 specifies the rules allowed in these profiles. An ontology with only sentences from Table 1 is *(i)* RL if it has no rules of Type (3); *(ii)* EL if it does not contain rules (1), (9), and (13); and *(iii)* QL if it does not contain rules (1), (4), (5), (7), (9)-(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 **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, and it is a *conjunctive query* (CQ) if it is $\vee$-free.

We consider two different semantics for query answering. Under the *classical semantics*, a tuple **t** of constants is an *answer* to $Q(\mathbf{x})$ w.r.t. an ontology $\mathcal{O}$ if $\mathcal{O} \models Q(\mathbf{t})$. Under the *active domain semantics*, **t** is an answer to Q w.r.t. $\mathcal{O}$ if there is a tuple $\mathbf{t}'$ of constants from $\mathcal{O}$ such that $\mathcal{O} \models \varphi(\mathbf{t}, \mathbf{t}')$, 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 **t**, a PEQ Q and an ontology $\mathcal{O}$ in a language $\mathcal{L}$, whether **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 [15]. 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 difference between both semantics manifests itself only in the presence of existentially quantified rules and queries; thus, both semantics coincide if either the input ontology is Datalog, or if all variables in the input query are free.

3. FACETED INTERFACES

In this section, we formalise a notion of a *faceted interface*, which provides a rigorous foundation for faceted search over RDF graphs enhanced with OWL 2 ontologies. To motivate our definitions, we will use an example based on an excerpt of DBpedia. Our goal is to find US presidents who graduated from Harvard or Georgetown and have a child who graduated from Stanford.

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

---

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

$$\text{President}(d_{tr}) \quad \text{President}(d_{bc}) \quad \text{Person}(d_{kr})$$
$$\text{Person}(d_{cc}) \quad \text{Country}(d_{us}) \quad \text{Country}(d_{uk})$$
$$\text{Univ}(d_h) \qquad\quad \text{Univ}(d_g) \qquad\quad \text{Univ}(d_s)$$

Specific information about documents is represented by literals. For example, Theodore Roosevelt's date of birth is encoded as dateOfBirth$(d_{tr}, 1858\text{-}10\text{-}27)$. *Most importantly, documents are also annotated with other documents; such annotations are represented in RDF and correspond to the following facts:*

$$\text{citiz}(d_{tr}, d_{us}) \;\; \text{citiz}(d_{bc}, d_{us}) \;\; \text{child}(d_{tr}, d_{kr}) \quad \text{child}(d_{bc}, d_{cc})$$
$$\text{grad}(d_{tr}, d_h) \;\; \text{grad}(d_{bc}, d_g) \;\; \text{grad}(d_{kr}, d_h) \qquad \text{grad}(d_{cc}, d_s)$$

Finally, DBpedia can be extended with ontological rules, which are exploited to describe the meaning of the predicates and constants in the vocabulary. Consider for example the rules given next:

$$\text{President}(x) \wedge \text{citiz}(x, d_{us}) \rightarrow \text{USpres}(x), \tag{1}$$
$$\text{USpres}(x) \rightarrow \text{President}(x) \wedge \text{citiz}(x, d_{us}), \tag{2}$$
$$\text{grad}(x, y) \rightarrow \text{Person}(x) \wedge \text{Univ}(y), \tag{3}$$
$$\text{Person}(x) \rightarrow \exists y.\big(\text{citiz}(x, y) \wedge \text{Country}(y)\big). \tag{4}$$

Rules (1) *and* (2) *define US presidents as those with US nationality. Rule* (3) *specifies the domain and range of* grad. *Finally,* (4) *mandates that each person has a (maybe unspecified) nationality.*

Analogously to traditional faceted search, we represent *facets* as pairs of a predicate (or facet name) and a set of values. In the context of RDF, however, documents can be used to annotate other documents, and thus annotations form a graph, rather than a tree. Thus, facet values can be either document URIs or literals. Examples of facet names are the relations 'grad' and 'dateOfBirth', and example values are documents such as 'd_s' and literals such as '1858-10-27'. 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 documents or literals. Finally, a special value any denotes the set of all values compatible with the facet name.

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 nonempty set, and either (i)* $X = $ type *and* $\Gamma \subseteq \mathbf{UP}$, *or (ii)* $X \in \mathbf{BP}$, any $\in \Gamma$ *and either* $\Gamma \subseteq \mathbf{C} \cup \{\text{any}\}$ *or* $\Gamma \subseteq \mathbf{UP} \cup \{\text{any}\}$. *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 name, *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.*

$F_1 = (\text{type}, \vee\{\text{USpres}, \text{Country}\})$,

$F_2 = (\text{child}, \vee\{\text{any}, d_{kr}, d_{cc}\})$, $F_3 = (\text{grad}, \vee\{\text{any}, d_h, d_s, d_g\})$,

$F_4 = (\text{citiz}, \wedge\{\text{any}, d_{us}, d_{uk}\})$, $F_5 = (\text{citiz}, \vee\{\text{any}, d_{us}, d_{uk}\})$.

The disjunctive facet F_1 *can be exploited to select the categories to which the relevant documents belong. Facet* F_2 *can be used to narrow down search results to those individuals with children; furthermore, the value* any *can be used to state that we are not looking for any specific child. The intuition behind* F_3, F_4, *and* F_5 *is similar; note, however, that facet* F_4 *is conjunctive.*

3.1 The Notion of Faceted Interface

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

Figure 1: A visualisation of the example interface

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 given by the following grammar, where* I_0 *and* $I_1 = (F, \Sigma)$ *are BFIs and* $F|_1 \in \mathbf{BP}$:

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

A BFI encodes user choices for a specific facet, e.g., the BFI $(F_1, \{\text{USpres}\})$ selects the documents categorised as US presidents. BFIs are put together in *paths*: sequences of nested facets that capture navigation between sets of documents. Documents are annotated with other documents 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 name. With nesting we can capture queries such as 'people with a child who graduated from Stanford' by using the interface $(F_2, \{\text{any}\})/(F_3, \{d_s\})$ which first selects people having (any) children and then those children with a Stanford degree. Finally, two types of branching can be applied: $(\text{path}_1 \wedge \text{path}_2)$ indicates that search results must satisfy the conditions specified by both path_1 and path_2, while $(\text{path}_1 \vee \text{path}_2)$ indicates that they must satisfy those in path_1 or path_2.

EXAMPLE 5. *Consider the following interface* I_{ex}, *which is visualised in our system as in Figure 1.*

$$\big((F_1, \{\text{USpres}\}) \wedge (F_3, \{d_h, d_g\})\big) \wedge \big((F_2, \{\text{any}\})/(F_3, \{d_s\})\big).$$

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.*

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* $w \in \{0, 1, \dots, 9, \cdot\}^*$ *be a variable. The query of* I *is the formula* $Q[I] = [\![I, x_\varepsilon, x_0]\!]$, *with one 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 name 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 $(\text{path}_1 \circ \text{path}_2)$ 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 $[\![\text{path}_i, v, x_w]\!] = \top(v)$, indicating that no value from the facets occurring in path_i is selected,

Basic Faceted Interfaces: $[\![(F,\Sigma),v,x_w]\!] =$	
$\top(v)$	if $\Sigma = \emptyset$
$\exists x_w \, F|_1(v,x_w)$	if any $\in \Sigma$
$\overset{\circ}{\underset{C\in\Sigma}{}} C(v)$	if $F|_1 = $ type and $\Sigma \neq \emptyset$
$\overset{\circ}{\underset{t_i\in\Sigma}{}} \exists x_{w\cdot i} \, F|_1(v,x_{w\cdot i}) \wedge x_{w\cdot i} \approx t_i$	if $F|_1 \neq$ type, any $\notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{C}$
$\overset{\circ}{\underset{C_i\in\Sigma}{}} \exists x_{w\cdot i} \, F|_1(v,x_{w\cdot i}) \wedge C_i(x_{w\cdot i})$	if $F|_1 \neq$ type, any $\notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{UP}$
Nesting: $[\![((F,\Sigma)/I),v,x_w]\!] =$	
$\top(v)$	if $\Sigma = \emptyset$
$\exists x_w \, F|_1(v,x_w) \wedge [\![I,x_w,x_{w\cdot 0}]\!]$	if any $\in \Sigma$
$\overset{\circ}{\underset{t_i\in\Sigma}{}} \exists x_{w\cdot i} \, F|_1(v,x_{w\cdot i}) \wedge$ $x_{w\cdot i} \approx t_i \wedge [\![I,x_{w\cdot i},x_{w\cdot i\cdot 0}]\!]$	if any $\notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{C}$
$\overset{\circ}{\underset{C_i\in\Sigma}{}} \exists x_{w\cdot i} \, F|_1(v,x_{w\cdot i}) \wedge$ $C_i(x_{w\cdot i}) \wedge [\![I,x_{w\cdot i},x_{w\cdot i\cdot 0}]\!]$	if any $\notin \Sigma$, $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{UP}$
Branching: $[\![(path_1 \circ path_2),v,x_w]\!] =$	
$([\![path_1,v,x_{w\cdot 0}]\!] \circ [\![path_2,v,x_{w\cdot 1}]\!])$	if $[\![path_1,v,x_{w\cdot 0}]\!] \neq \top(v)$ $[\![path_2,v,x_{w\cdot 1}]\!] \neq \top(v)$
$[\![path_1,v,x_{w\cdot 0}]\!]$	if $[\![path_1,v,x_{w\cdot 0}]\!] \neq \top(v)$ $[\![path_2,v,x_{w\cdot 1}]\!] = \top(v)$
$[\![path_2,v,x_{w\cdot 1}]\!]$	if $[\![path_1,v,x_{w\cdot 0}]\!] = \top(v)$ $[\![path_2,v,x_{w\cdot 1}]\!] \neq \top(v)$
$\top(v)$	if $[\![path_1,v,x_{w\cdot 0}]\!] = \top(v)$ $[\![path_2,v,x_{w\cdot 1}]\!] = \top(v)$

Table 2: Semantics of faceted interfaces

then $path_i$ is ignored. Finally, nesting involves a "shift" of variable from the parent BFI to the nested subexpression.

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

$$Q_{\mathsf{ex}}(x) = \mathsf{USpres}(x) \wedge (\exists y_1 \, (\mathsf{grad}(x,y_1) \wedge y_1 \approx d_h)$$
$$\vee \, \exists y_2 \, (\mathsf{grad}(x,y_2) \wedge y_2 \approx d_g))$$
$$\wedge \, \exists z \, (\mathsf{child}(x,z) \wedge \exists w(\mathsf{grad}(z,w) \wedge w \approx d_s)).$$

If we consider only facts, the answer is empty (e.g., no document is categorised as 'US president'). If we also consider the ontology rules, however, we obtain d_{bc} (i.e., Bill Clinton) as an answer.

Our notion of interface motivates the class of *faceted queries*, i.e., PEQs that can be captured by some faceted interface.

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.*

Note that our notion of interface abstracts from several considerations that are critical to GUI design. For instance, our notion 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 Interfaces with Refocussing

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 *refocussing* (or *pivoting*) functionality [9, 10]. We next extend faceted interfaces to allow for such functionality.

DEFINITION 9. *Let* focus *be a symbol not occurring in $\mathbf{C} \cup \mathbf{UP} \cup \mathbf{BP}$. An* extended basic faceted interface *(EBFI) is either*

Extended Basic Faceted Interfaces: $[\![(F,\Sigma \cup \{\mathsf{focus}\}),v,x_w]\!] =$	
$F|_1(v,x_w)$	if $\Sigma = \emptyset$
$[\![(F,\{\mathsf{focus}\}),v,x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{C} \cup \{\mathsf{any}\}$
$[\![((F,\{\mathsf{focus}\})/((\mathsf{type},\vee F|_2),\Sigma)),v,x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{UP} \cup \{\mathsf{any}\}$
Nesting: $[\![((F,\Sigma \cup \{\mathsf{focus}\})/I),v,x_w]\!] =$	
$F|_1(v,x_w) \wedge [\![I,x_w,x_{w\cdot 0}]\!]$	if $\Sigma = \emptyset$
$[\![((F,\{\mathsf{focus}\})/I),v,x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{C} \cup \{\mathsf{any}\}$
$[\![((F,\{\mathsf{focus}\})/(((\mathsf{type},\vee F|_2),\Sigma) \wedge I)),v,x_w]\!]$	if $\Sigma \neq \emptyset$ and $\Sigma \subseteq \mathbf{UP} \cup \{\mathsf{any}\}$

Table 3: Semantics of extended faceted interfaces

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

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

DEFINITION 10. *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:*

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

Finally, 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.*

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

$$((F_1,\{\mathsf{USpres}\}) \wedge (F_3,\{d_h,d_g\})) \wedge ((F_2,\{\mathsf{focus}\})/(F_3,\{d_s\})).$$

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 the existential quantifier $\exists x$ to the resulting query, thus obtaining $Q'_{\mathsf{ex}}(z)$:

$$\exists x \, (\mathsf{USpres}(x) \wedge (\exists y_1 \, (\mathsf{grad}(x,y_1) \wedge y_1 \approx d_h)$$
$$\vee \, \exists y_2 \, (\mathsf{grad}(x,y_2) \wedge y_2 \approx d_g))$$
$$\wedge \, (\mathsf{child}(x,z) \wedge \exists w(\mathsf{grad}(z,w) \wedge w \approx d_s))).$$

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

4. ANSWERING FACETED QUERIES

Each time a user selects a facet value to refine the search results, a faceted search system must compute the answers to a query. 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.

Our main result is that, in contrast to PEQs (and even CQs), faceted query evaluation over datasets is tractable; furthermore, the problem remains tractable in most cases if we consider ontologies belonging to the OWL 2 profiles. Our tractability results concern

Algorithm 1: ANSWER-FQ: Faceted Queries over Datasets

> **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$(\varphi'_i, \mathcal{D})$
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** REWRITE$(\varphi_1) \wedge$ REWRITE(φ_2)
4 **case** $\varphi = \varphi_1 \vee \varphi_2$ **return** $C_{\varphi_1 \vee \varphi_2}(y)$ with $y = \mathsf{fvar}(\varphi_i)$

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

The rationale behind our tractability result is that PEQs originating from faceted interfaces are of a rather restricted shape, which is determined by Table 2 in Section 3. A closer look at the table reveals 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.

DEFINITION 11. *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 $R(y, y')$.*

Note that query $Q_{\mathsf{ex}}(x)$ in Example 7 is tree-shaped. The second 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 summarised as follows:

PROPOSITION 12. *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 x.*

We next show how the restricted shape of faceted queries can be exploited to make query answering more efficient. We start by providing a polynomial algorithm for answering faceted queries over datasets. 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 dataset 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 uniquely associated to φ.

EXAMPLE 13. *Query Q_{ex} 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 B. Clinton, T. Roosevelt and K. Roosevelt as answers. Then, extend the dataset with facts $C_\varphi(d_{bc})$, $C_\varphi(d_{tr})$ and $C_\varphi(d_{kr})$ 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 document is explicitly categorised as US president.*

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

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

Thus, faceted queries can be evaluated in polynomial time with an oracle for the evaluation of tree-shaped CQs. By a classic result, acyclic CQs (and hence also tree-shaped CQs as in Def. 11) can be answered in polynomial time [16]. Thus, tractability tree-shaped CQ evaluation transfers to the evaluation of faceted queries.

COROLLARY 15. *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 [15] 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 first computes the dataset $\mathcal{D}$ of all facts entailed by $\mathcal{O}$ and then answers Q w.r.t. the dataset $\mathcal{D}$. The correctness of Algorithm 2 follows directly from Theorem 14 and the following proposition.

PROPOSITION 16. *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 active domain answers to Q w.r.t. $\mathcal{O}$ and w.r.t. $\mathcal{D}$ coincide.*

Fact entailment is tractable for all the OWL 2 profiles; thus, by committing to the active domain semantics of SPARQL we can achieve tractability without emasculating the ontology language.

THEOREM 17. *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.*

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 query answering under classical semantics if the input ontology is Datalog. An immediate consequence is that our results in Theorem 17 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 query answering [17, 18]. These techniques are currently applicable to *guarded* EL ontologies, i.e., EL ontologies without axioms of Type (5). The idea is to rewrite rules of Type (3) in Table 1 into Datalog by Skolemising existentially quantified variables into fresh constants.

DEFINITION 18. *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 $A(x) \rightarrow P(x, c_{R,B}), P(x, y) \rightarrow R(x, y), P(x, y) \rightarrow B(y)$, 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 facts [17]. The following theorem establishes that the evaluation of faceted queries is also preserved.

THEOREM 19. *Let Q be a faceted query, $\mathcal{O}$ a guarded EL ontology, and c a constant in $\mathcal{O}$. Then, $\mathcal{O} \models Q(c)$ iff $\Xi(\mathcal{O}) \models Q(c)$.*

Thus, we can answer faceted queries over an EL ontology $\mathcal{O}$ by applying Algorithm 2 to $\Xi(\mathcal{O})$. Since Ξ is a linear transformation and $\Xi(\mathcal{O})$ is an RL ontology, we can conclude tractability of faceted query evaluation for EL (a result consistent with existing results for acyclic CQs in EL [19]). In contrast, the evaluation of acyclic CQs is already NP-hard for OWL 2 QL [20] and we can show that faceted query evaluation is NP-complete for OWL 2 QL. The following theorem summarises our results.

THEOREM 20. *Faceted query evaluation under classical semantics is (i) PTIME-complete for RL and guarded EL ontologies; and (ii) NP-complete for QL ontologies.*

4.4 Extended Faceted Queries

We conclude by arguing that the refocussing functionality does not increase complexity of query evaluation. PEQs obtained from EFIs satisfy Proposition 12, 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 15 also holds for extended faceted queries. From this, and using the same techniques as in the proofs of Theorems 17 and 20, we obtain that:

PROPOSITION 21. *Extended faceted query evaluation under classical semantics is (i) PTIME-complete for RL and guarded 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.*

5. INTERFACE GENERATION & UPDATE

Faceted navigation is an interactive process. Starting with an initial interface generated from a keyword search, users 'tick' or 'untick' facet values and the system reacts by updating both search results (query answers) and facets available for further navigation.

EXAMPLE 22. *Consider the interactive construction of our example interface I_{ex}. Navigation starts with the following 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:

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

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:

$$I_2 = (F_1, \{\text{USpres}\}) \wedge (F_3, \{d_h, d_g\}) \wedge (F_2, \emptyset).$$

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

$$I_3 = (F_1, \{\text{USpres}\}) \wedge (F_3, \{d_h, d_g\}) \wedge \big((F_2, \{\text{any}\})/(F_3, \emptyset)\big).$$

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 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 applications, however, different values in a facet are typically 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 23. *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 what we call a *facet graph*. The graph can be seen as a concise representation of $\mathcal{O}$, and 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 names (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 $X = \text{type}$, and (the constant) v is entailed to be an instance of (the unary predicate) w.

DEFINITION 24. *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

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

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

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

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

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

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

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

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

It follows from the following proposition that facet graph computation can be efficiently implemented. In practice, the graph can be precomputed when first loading data and ontology, stored in RDF, and accessed using SPARQL queries. In this way, reasoning tasks associated to faceted search are performed offline.

PROPOSITION 26. *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.*

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 27. *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 an interface to a facet graph.

DEFINITION 28. *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 holds: (i) there is an X_2-labelled edge from v_1 to v_2; or (ii) $v_1 =$ any and*

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

there is an X_2-labelled edge from some $u \in \text{range}_G(X_1)$ to v_2; or (iii) $v_2 =$ any and v_1 has an outgoing X_2-edge; or (iv) $v_1 = v_2 =$ 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, one of the following holds: (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 name 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 name 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 name X_2.

5.2 Interface Generation

Algorithm 3 shows how a fresh interface can be generated from a starting set mS 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 name 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 backend would first compute an initial set D of documents 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 29. *On input G and S, Algorithm 3 outputs a simple interface that conforms to G.*

5.3 Interface Update

The initial interface where no facet value has been yet selected marks the start of the navigation process. User actions on an interface can be seen as elementary 'ticking' and 'unticking' operations on facet values that result in another interface. We define these actions by exploiting the tree representation of interfaces (c.f. Definition 27). We start with the ticking operation.

DEFINITION 30. *The action* TICK *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 4: TICK

> **INPUT** : $I, (w, v)$, and G as in Def. 30, with $\lambda(w) = (F, \Sigma)$
> **OUTPUT** : A simple EFI

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

Algorithm 5: UNTICK

> **INPUT** : $I, (w, v)$ and G as in Def. 32, 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      $S := \{(w', v') \mid (w', v')$ is uniquely justified by (w, v) in G, $\lambda(w') = (F', \Sigma')$ and $v' \in \Sigma\}$
4   **for each** $(w', v') \in S$ **do** $I := \text{UNTICK}(I, (w', v'), G)$
5      $I_{out} :=$ replace Σ in I with $\Sigma \setminus \{v\}$
6 $\lambda_{out} :=$ labelling function of $\text{tree}(I_{out})$
7 **for each** *node* w' *in* $\text{tree}(I_{out})$ **do**
8      $(F', \Sigma') := \lambda_{out}(w')$
9   **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}

Algorithm TICK 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. Finally, Algorithm TICK 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 31. *Assume that $I, (w, v)$ and G are as in Definition 30. If I conforms to G, then $\text{TICK}(I, (w, v), G)$ is a simple EFI that also conforms to G.*

We next define the unticking operation. Intuitively, when unticking a value v in a given position of an interface all values that were justified by v (and only by v) should also be unticked. 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 32. *The action* UNTICK *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 $\text{tree}(I)$. The result is the interface computed by Algorithm 5 .*

Algorithm UNTICK considers two cases depending on what kind of value v is unticked. 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 unticked (Lines 2-5). Unticking 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' with the rest (Lines 7-9).

PROPOSITION 33. *Assume that $I, (w, v)$ and G are as in Definition 32. If I conforms to G, then* $\text{UNTICK}(I, (w, v), G)$ *is a simple EFI that also conforms to G.*

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 34. *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.* TICK *is applicable to $I, (w, v)$ and G, the following holds: (i) $Q[\text{TICK}(I, (w, v), G)]$ has non empty answers w.r.t. $\mathcal{O}$; and (ii) the answers to $Q[\text{TICK}(I, (w, v), G)]$ w.r.t. $\mathcal{O}$ are different from the answers to $Q[I]$ w.r.t. $\mathcal{O}$.*

Note that the transition from interface I_0 to I_1 in Example 22 involves a minimisation step. The BFI in I_0 involving F_5 is pruned since ticking a value will either not affect the search results (if any or d_{us} is ticked) or yield an empty set of answers (if d_{uk} is ticked).

To avoid overwhelming users with irrelevant information, our system minimises the output of Alg. 4 before showing it to the user. Our system 'runs' each possible expansion of a EFI in the background by calling the reasoning engine, and prunes all facet values that do not change query answers, or make them empty.

6. IMPLEMENTATION AND TESTING

We have developed a faceted search platform providing the following main functionality: *(i)* computation of facet graphs from an ontology; *(ii)* interface generation from facet graphs; and *(iii)* interface update in response to user actions. Our platform relies on an external triple store for querying and OWL reasoning.

In our implementation, facet graphs are represented in RDF: each R-labelled edge from v to w is stored as a triple (v, R, w). A graph G can be either loaded from an existing RDF document, or constructed from (the facts entailed by) the ontology $\mathcal{O}$ by adding additional edges. The kinds of relevant additional edges are described by means of customisable rules, and the edges themselves are computed by materialisation of such rules. Furthermore, our platform implements Algorithm 3 for generating interfaces, Algorithms 4 and 5 for interface update, and strategies for interface minimisation. Our algorithms can operate both under the assumption that the facet graph G is explicitly materialised, or it is defined virtually using rules and then generated 'on the fly' as needed.

We have implemented a proof-of-concept system, called SemFacet, that bundles our platform with JRDFox[3] as triple store, Lucene for keyword search, and an HTML 5 GUI [21]. The system's architecture is given in Figure 2(c). Our system is available online.[4]

6.1 Performance Metrics

Performance of our platform 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: (i) $t[\textit{run query}]$: time to execute an atomic query; and (ii) $t[\textit{look up}]$: time to iterate over query results.

Interface generation (Algorithm 3) requires computing all triples (v, w, u) in the facet graph G for each v in the input nodes S, and then iterating over the results to compose the interface. Thus, to estimate the cost of interface generation (t_{CI}), we can use Alg. 6

---

[3] www.cs.ox.ac.uk/isg/tools/RDFox/

[4] http://www.cs.ox.ac.uk/isg/tools/SemFacet/

#(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
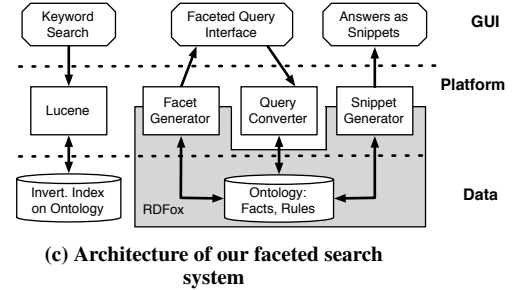
(c) Architecture of our faceted search system

Figure 2: Experimental results for JRDFox, Stardog, and Sesame

---

Algorithm 6: CREATEINTERFACEIMPLEMENTED

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 $\quad$ Triples$_v :=$ SELECT ?y,?z FROM G WHERE $(v, ?y, ?z)$
4 $\quad$ **for each** $t \in$ Triples$_v$ **do** $I :=$ COMPOSEINTERFACE(t, I)

---

instead of Alg. 3. We assume constant time for the call to COMPOSEINTERFACE. The cost can then be estimated as follows:

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

In this expression, $\#[answers]$ is the union of all sets Triples$_v$ for each $v \in S$. In the worst-case, $\#[answers]$ is $|G|$, whereas in the best-case it corresponds to $|S|$. For improved efficiency, our platform implements a variation of Algorithm 6 where facets are computed lazily: facet names 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. Then, $\#[answers]$ is estimated as follows, where the number of facet names corresponds to the number of different edge labels in G, and the number of facet values to the number of nodes:

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

The cost t_{CI} in (5) can also be used to estimate the cost of interface updates. The Algorithm for ticking (Sec. 5.3) can be seen as a variant of Alg. 6 with S the set of values relevant to the tick. In the case of unticking, the worst-case cost 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 UNTICK (Alg. 5), whereas t_{CI} estimates the cost of a single recursive call.

6.2 Performance Estimations

To estimate the parameters $t[run\ query]$ and $t[look\ up]$, thus also estimating the cost t_{CI} of interface generation, we have conducted experiments over a fragment of DBpedia enriched with OWL 2 RL rules and we have used JRDFox, Stardog (http://stardog.com/) and Sesame (http://www.openrdf.org/). All experiments were conducted on a MacBook Pro laptop with OS X 10.8.5, 2.4 GHz Intel Core i5 processor, and 8GB 1333 MHz DDR3 memory. Since triple stores such as JRDFox 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 using 8GB of RAM. Each experiment was executed 100 times it total, and 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.

Results are summarised in Figures 2(a) and 2(b). Figure 2(a) estimates $\#[answers] \times t[look\ up]$ by measuring time required to iterate over an answer set of a given size. In turn, Figure 2(b) estimates $|S| \times t[run\ query]$ by computing the times required for the triple store to answer a given number of atomic queries. We can

make the following observations: (i) The time needed to iterate over query results is small in comparison to query execution times; for example, to execute $10,000$ queries, JRDFox requires $0.498s$, whereas to iterate over $10,000$ answers it requires $0.002s$. This should be taken into account when optimising interface generation. (ii) 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. As shown in Figure 2(b), obtaining a reasonable cap is important since query execution is expensive. For example with a cap of $1,000$ 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 $0.2s$ for 1 million results and $2s$ for 10 million. We were not even able to conduct experiments with 10 million answers over Stardog and Sesame since loading the data in our machine consumed all RAM and system behavior became unstable. The facet graph for the whole of DBpedia contains 24 million facet values and $1,843$ facet names [4]. JRDFox would require $5s$ 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 names instantaneously.

7. RELATED WORK

The design of visual interfaces for querying ontologies has received significant attention in recent years. Existing systems typically support query formulation by exploiting either a form of controlled natural language (e.g., Quelo [22]), or different graphical representations for queries (e.g., SEWASIE [23], iSPARQL [24], OntoVQL [25], Wonder [26], or the OptiqueVQS [27], or other approaches, including interactive exploration of [28].

Faceted search over RDF has also attracted a great deal of attention. Developed systems include mSpace [5], /facet [7], Piggy Bank [8], Tabulator [2], gFacet [6], Humboldt [9], Parallax [10], Longwell [29], faceted DBpedia [4], X-ENS [3], Broccoli [30], and others [31–33]. The functionalities provided by these systems include navigation through different sets of documents, refocussing, and interface minimisation via elimination of dead-ends.

These works are primarily systems-oriented and their main focus is on improving user experience, development of ranking functions and value grouping heuristics [11, 13], and backend optimisation via indexing schemes [4, 34]. Our framework was inspired by the capabilities of existing systems, and covers their main functionalities. Since our aim was to study the fundamental properties of query languages and update tasks, our framework abstracts from (and is compatible with) usability, ranking, and indexing considerations.

The expressivity of the query languages supported by existing systems is discussed in the literature mostly verbally, which makes it difficult to determine the underpinning SPARQL fragment. Most systems seem to support some form of conjunctive queries (e.g.,

see [11, 13]), and disjunction is present only in a limited form [3, 4]. The approach of [12] allows for conjunction, disjunction, and other operators, e.g., negation, thus, they cover a wider fragment of SPARQL than we do. At the same time, [12] is orthogonal to other faceted search approaches for RDF, including ours: their facet values are possible *queries* rather than (set of) documents, and a selection of a facet value corresponds to a syntactic query transformation rather than to setting a filter on a set of documents. Expressiveness of this approach is determined by the expressiveness of queries that are allowed to be used as facet values.

When query languages have not been formalised, the complexity of query answering was not addressed. The common assumption is that user selections in an interface are compiled in SPARQL [12] or Prolog [7], and executed by a query evaluation engine over the underlying RDF data. Complexity considerations are, however, critical when RDF data is enhanced with OWL 2 reasoning. This setting was not addressed by existing systems, where ontological axioms are limited to class and property hierarchies [7, 11], and reasoning plays little or no role. Interface generation and update mechanisms are mostly informally described. A common approach is to generate and update interfaces from the RDF data graphs. Since we generate interfaces from facet graphs that subsume RDF datasets, we see our approach as a generalisation of existing work. Finally, scalability of faceted search systems over large RDF datasets is an important concern [4, 34]. Since facet graphs can be much larger than the underlying RDF datasets, scalability becomes even more critical in our setting. Our experiments, however, suggest that our approach is feasible in practice.

The works closest to ours are [11–13]. The query language in [13] is formalised using CQs, whereas the language in [11] (also conjunctive) is introduced via set operations. These works, however, do not study the complexity of query answering, and ontological reasoning is also not considered. We can also find notions of facet trees and graphs in the literature [7, 11, 13, 29, 35]. These represent combinations of (possibly nested) facets displayed in a GUI as a tree or a graph, and they depend on both search results and front-end considerations. We see our notions of interface and facet graph as GUI-independent generalisation of existing notions since our graphs are derived from ontologies and independently from search results. Finally, the 'navigation graph' of [12] defines navigation links at the syntactic level as query transformations, rather than semantic relations between sets and objects, as in our case.

8. CONCLUSION AND FUTURE WORK

In this paper, we have established theoretical foundations for faceted search in the context of RDF and OWL 2. Our results suggest many problems for future work, such as exploring extensions of our update algorithms beyond simple interfaces. Concerning system design, substantial work is needed to improve GUI design, especially with respect to refocussing. We are also planning to benchmark our platform on real-world applications.

9. REFERENCES

[1] D. Tunkelang. *Faceted Search*. Morgan & Claypool Publishers, 2009.
[2] T. Berners-Lee, J. Hollenbach, K. Lu, J. Presbrey, E. Prudhommeaux, and M. M. C. Schraefel. Tabulator Redux: Browsing and Writing Linked Data. In: *LDOW*. 2008.
[3] P. Fafalios and Y. Tzitzikas. X-ENS: Semantic Enrichment of Web Search Results at Real-Time. In: *SIGIR*. 2013.
[4] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, M. Bürgle, et al. Faceted Wikipedia Search. In: *BIS*. 2010.
[5] m.c. schraefel, D. A. Smith, A. Owens, A. Russell, C. Harris, and M. L. Wilson. The Evolving mSpace Platform: Leveraging the Semantic Web on the Trail of the Memex. In: *Hypertext*. 2005.
[6] P. Heim, J. Ziegler, and S. Lohmann. gFacet: A Browser for the Web of Data. In: *IMC-SSW*. 2008.
[7] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A Browser for Heterogeneous Semantic Web Repositories. In: *ISWC*. 2006.
[8] D. Huynh, S. Mazzocchi, and D. R. Karger. Piggy Bank: Experience the Semantic Web Inside Your Web Browser. In: *J. Web Sem.* 5.1 (2007).
[9] G. Kobilarov and I. Dickinson. Humboldt: Exploring Linked Data. In: *LDOW*. 2008.
[10] D. F. Huynh and D. R. Karger. Parallax and Companion: Set-based Browsing for the Data Web. 2013.
[11] E. Oren, R. Delbru, and S. Decker. Extending Faceted Navigation for RDF Data. In: *ISWC*. 2006.
[12] S. Ferré and A. Hermann. Semantic Search: Reconciling Expressive Querying and Exploratory Search. In: *ISWC*. 2011.
[13] A. Wagner, G. Ladwig, and T. Tran. Browsing-oriented Semantic Faceted Search. In: *DEXA*. 2011.
[14] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles. In: *W3C Recommendation* (2009).
[15] *W3C: SPARQL 1.1 Entailment Regimes*. www.w3.org/TR/sparql11-entailment/.
[16] M. Yannakakis. Algorithms for Acyclic Database Schemes. In: *VLDB*. 1981.
[17] G. Stefanoni, B. Motik, and I. Horrocks. Introducing Nominals to the Combined Query Answering Approaches for EL. In: *AAAI*. 2013.
[18] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The Combined Approach to Ontology-Based Data Access. In: *IJCAI*. 2011.
[19] M. Bienvenu, M. Ortiz, M. Simkus, and G. Xiao. Tractable Queries for Lightweight Description Logics. In: *IJCAI*. 2013.
[20] S. Kikot, R. Kontchakov, and M. Zakharyaschev. On (In)Tractability of OBDA with OWL 2 QL. In: *DL*. 2011.
[21] M. Arenas, B. Cuenca Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, and E. Jiménez-Ruiz. SemFacet: Semantic Faceted Search over Yago. In: *WWW*. 2014.
[22] E. Franconi, P. Guagliardo, M. Trevisan, and S. Tessaris. Quelo: an Ontology-Driven Query Interface. In: *DL*. 2011.
[23] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. The SEWASIE Network of Mediator Agents for Semantic Search. In: *J. UCS* 13.12 (2007).
[24] *iSPARQL QBE*. http://dbpedia.org/isparql/.
[25] A. Fadhil and V. Haarslev. OntoVQL: A Graphical Query Language for OWL Ontologies. In: *DL*. 2007.
[26] D. Calvanese, C. M. Keet, W. Nutt, M. Rodriguez-Muro, and G. Stefanoni. Web-based Graphical Querying of Databases Through an Ontology: the Wonder System. In: *SAC*. 2010.
[27] A. Soylu, M. Giese, E. Jiménez-Ruiz, E. Kharlamov, D. Zheleznyakov, and I. Horrocks. OptiqueVQS: Towards an Ontology-based Visual Query System for Big Data. In: *MEDES*. 2013.
[28] N. Manolis and Y. Tzitzikas. Interactive Exploration of Fuzzy RDF Knowledge Bases. In: *ESWC (1)*. 2011.
[29] C. Veres, K. Johansen, and A. L. Opdahl. Browsing and Visualizing Semantically Enriched Information Resources. In: *CISIS*. 2010.
[30] H. Bast, F. Bäurle, B. Buchhold, and E. Haußmann. Easy Access to the Freebase Dataset. In: *WWW*. 2014.
[31] O. Suominen, K. Viljanen, and E. Hyvönen. User-Centric Faceted Search for Semantic Portals. In: *ESWC*. 2007.
[32] P. Haase, D. M. Herzig, M. A. Musen, and T. Tran. Semantic Wiki Search. In: *ESWC*. 2009.
[33] S. Buschbeck, A. Jameson, R. Troncy, H. Khrouf, O. Suominen, and A. Spirescu. A Demonstrator for Parallel Faceted Browsing. In: *EKAW*. 2012.
[34] H. Bast and B. Buchhold. An Index for Efficient Semantic Full-Text Search. In: *CIKM*. 2013.
[35] P. Heim, T. Ertl, and J. Ziegler. Facet Graphs: Complex Semantic Querying Made Easy. In: *ESWC*. 2010.