Enabling Faceted Search over OWL 2 with SemFacet*

Marcelo Arenas[1], Bernardo Cuenca Grau[2], Evgeny Kharlamov[2],
Šarūnas Marciuška[2], and Dmitriy Zheleznyakov[2]

[1] Pontificia Universidad Católica de Chile; [2] University of Oxford

Abstract. Lots of applications nowadays rely on RDF, OWL 2, and SPARQL 1.1 for storing, publishing and querying data. However, SPARQL 1.1 is not targeted towards end-users, and suitable query interfaces are needed. Faceted search is a prominent approach to facilitate end-users query formulation which is widely used in information systems. This approach was recently adapted to the context of RDF, however, the proposed solutions lack rigorous theoretical underpinning and essentially ignore OWL 2 axioms. We develop a clean theoretical framework for faceted search that accounts for both RDF data and OWL 2 axioms. We implemented and tested some of our solutions in the SemFacet platform.

1 Introduction

In the last decade we have witnessed a constant increase in the number of applications that store and publish data in RDF, in the of use OWL 2 ontologies for providing background knowledge about the application domain and enriching query answers with information not explicitly given in the data, and in the use of SPARQL 1.1 for querying this data. It was acknowledged by many that writing SPARQL 1.1 queries requires special training and is not well-suited for the majority of end users. Thus, an important challenge is the development of simple yet powerful query-formulation interfaces that capture well-defined fragments of SPARQL 1.1.

This challenge was acknowledged by the community, and a number of approaches to facilitate query formulation over RDF and OWL 2 have been proposed in the last decade. Proposed solutions rely on different query formulation paradigms and include controlled natural language, e.g., [4–8], diagram based approaches, e.g., [9–15], faceted search interfaces,e.g., [16–18], and others. In this work we focus on faceted search due to its importance in Web bases information systems and intuitiveness to end users.

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* [19]. 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 RDF-based faceted search systems have been developed, e.g. [16–18, 20–26]. Although there has been intensive efforts in system development, the theoretical underpinnings received less attention [27–29].

* Work supported by the Royal Society, the EPSRC projects Score!, Exoda, and MaSI[3], and the FP7 project OPTIQUE [1–3] under the grant agreement 318338.

[1] We use the term 'document' to refer to any resource that can be referenced using a URI.

In particular, it is unclear what fragments of SPARQL 1.1 can be naturally captured using faceted search as a query paradigm, and what is the complexity of answering such queries. Moreover, faceted search interfaces are typically generated and updated based only on RDF data graphs. We see this as an important limitation as OWL 2 axioms are essentially ignored by the existing solutions, thus overlooking that ontological axioms can be used to enrich query answers with implicit information. Besides, these axioms provide schema-level structure that can be exploited to improve faceted interfaces. Finally, purely RDF-based faceted search systems are data-centric, and hence cannot be exploited to browse large ontologies or to pose meaningful queries at the schema level.

We address these limitations of existing solutions by formalising 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. For faceted queries, i.e., queries that can be encoded by faceted interfaces, we study the expressivity and complexity of evaluation over RDF data enhanced with OWL 2 axioms. Finally, we study interface generation and update. Existing techniques for RDF are based on exploration of the underlying RDF graph: 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 special graph-based representation of OWL 2 ontologies and their logical entailments for the purpose of faceted navigation. Further details on our techniques can be found in [30].

To put our ideas in practice we developed a faceted search platform, called SemFacet (available for download and installation, and as a Web service [31, 32]), for generating and updating faceted interfaces. SemFacet relies on an external triple store with OWL 2 reasoning capabilities, and it is compatible with faceted search GUIs and text search engines for retrieving documents from keywords. For the demonstration purpose we bundled SemFacet with the triple store JRDFox, the search engine Lucene, and our own faceted search GUI. We have tested SemFacet over synthetic ontologies as well as Yago [33] extended with DBpedia (`dbpedia.org/`) abstracts with encouraging results.

2 Preliminaries

We use standard notions from first-order logic. We assume pairwise disjoint infinite sets of constants, unary predicates, and binary predicates. 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 $\mathbf{x}$, $\mathbf{z}$, and $\mathbf{y}$ are pairwise disjoint tuples of variables, $\varphi(\mathbf{x}, \mathbf{z})$ is a conjunction of atoms with variables in $\mathbf{x} \cup \mathbf{z}$, and $\exists \mathbf{y} \, \psi(\mathbf{x}, \mathbf{y})$ 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 $\exists \mathbf{y} \, \psi(\mathbf{x}, \mathbf{y})$ has at most one atom and all variables are universally quantified. OWL 2 defines three *profiles*: RL, EL, and QL, weaker languages with favourable computational properties [34]. Every profile ontology can be normalised as a set of rules and

facts using the correspondence of OWL 2 axioms with first-order logic and a variant of the structural transformation (e.g., see [35]). Thus, we define an *ontology* $\mathcal{O}$ as a finite set of rules and facts. We refer to [30, 35] for the details of rules corresponding to the OWL 2 profiles. A *positive existential query* (PEQ) is a formula that may have free variables which is constructed using $\wedge$, $\vee$ and $\exists$. A PEQ s *monadic* if it has one free variable, and it is a *conjunctive query* (CQ) if it is $\vee$-free.

We consider two different semantics for query answering. Under the *classical semantics*, given a query $Q(\mathbf{x}) = \exists \mathbf{y} \varphi(\mathbf{x}, \mathbf{y})$, we have that a tuple $\mathbf{t}$ of constants is an *answer* to $Q(\mathbf{x})$ w.r.t. an ontology $\mathcal{O}$ if $\mathcal{O} \models \exists \mathbf{y} \varphi(\mathbf{t}, \mathbf{y})$. 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 $\mathcal{O}$ such that $\mathcal{O} \models \varphi(\mathbf{t}, \mathbf{t}')$. The evaluation problem under the 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 classical (resp. active domain) semantics. The classical semantics is the default in first-order logic, whereas active domain is the default semantics of the SPARQL 1.1 entailment regimes [36]. 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

Our notion of *faceted interface* comes with a clean first-order logic semantics, and provides a rigorous foundation for faceted search over RDF graphs enhanced with OWL 2 ontologies. We start with an example based on an excerpt of from Yago [33]. Our goal is to find presidents who graduated from St. Petersburg or Georgetown and have a child who graduated from some university.

Example 1. The document d_{vp} and d_{bc} for Vladimir Putin and Bill Clinton are annotated with the category 'president', and d_{vp} is also annotated with 'Russian'. Putin's daughter Yekaterina d_{yp} and Clinton's daughter Chelsea d_{cc} are categorised as 'person'. The documents for St. Petersburg State Uni. d_{sp}, Stanford Uni. d_s, and Georgetown Uni. d_g are categorised under 'university'. These annotations are given in RDF and correspond to the following facts:

$$\begin{array}{llll} \mathsf{President}(d_{vp}), & \mathsf{President}(d_{bc}), & \mathsf{Russian}(d_{vp}), & \mathsf{Person}(d_{yp}), \\ \mathsf{Person}(d_{cc}), & \mathsf{Univ}(d_{sp}), & \mathsf{Univ}(d_s), & \mathsf{Univ}(d_g). \end{array}$$

In RDF specific information about documents is represented using literals, e.g., Vladimir Putin's date of birth is encoded as $\mathsf{dateOfBirth}(d_{vp}, \text{1952-10-07})$. Most importantly, documents are also annotated with other documents:

$$\mathsf{child}(d_{vp}, d_{yp}),\ \mathsf{child}(d_{bc}, d_{cc}),\ \mathsf{grad}(d_{vp}, d_{sp}),\ \mathsf{grad}(d_{bc}, d_g),\ \mathsf{grad}(d_{cc}, d_s).$$

Moreover, Yago can be extended with ontological rules. Consider for example the following rule that says that at least one child of each Russian president graduate from some university, which is a reasonable assumption:

$$\mathsf{Russian}(x) \wedge \mathsf{President}(x) \wedge \mathsf{child}(x, y) \rightarrow \mathsf{child}(x, z) \wedge \mathsf{grad}(z, w) \wedge \mathsf{Univ}(w). \quad (1)$$

Analogously to traditional faceted search, we represent a *facet* as a pair of a predicate (or facet name) and a set of values. In the context of RDF, however, documents (and not just strings) can be used to annotate other documents, and thus annotations form a graph, rather than a tree. Consequently, 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' (Stanford) and literals such as '1952-10-07'. Selection of multiple values within a facet can be interpreted conjunctively or disjunctively, and hence we distinguish between conjunctive and disjunctive facets. Furthermore, we distinguish a special facet type, whose values are categories (i.e., unary predicates) rather than specific documents or literals. Finally, a special value any denotes the set of all values compatible with the facet name.

Definition 1. *Let* type *and* any *be special symbols. 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* Γ *is a set of unary predicates, or*

(ii) X *is a binary predicate and* Γ *is a set that consists of* any *and either some constants or some unary predicates.*

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 2. Consider the following facets: $F_1 = (\mathsf{type}, \vee\{\mathsf{President}, \mathsf{Univ}\})$, $F_2 = (\mathsf{child}, \vee\{\mathsf{any}, d_{yp}, d_{cc}\})$ and $F_3 = (\mathsf{grad}, \vee\{\mathsf{any}, d_{sp}, d_g, d_s\})$. The disjunctive facet F_1 can be exploited to select the categories to which the relevant documents belong (president or university). Facet F_2 can be used to narrow down search results to those individuals with children d_{yp} or d_{cc}; furthermore, the value any can be used to state that we are not looking for any specific child. F_3 allows to select graduation places. □

3.1 The Notion of Faceted Interface

Our faceted interfaces encode both a query (whose answers determine the search results) and the choices of facet values available for further refinement.

Definition 2. *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* $F|_1 \in \mathbf{BP}$*:*

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

A BFI encodes user choices for a specific facet, e.g., the BFI $(F_1, \{\mathsf{President}\})$ selects the documents categorised as 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 $(F_2, \{\mathsf{any}\})/(F_3, \{d_s\})$ which first selects people having (any) children and then those children with a Stanford degree. Finally, two types of branching can be applied: $(\mathit{path}_1 \wedge \mathit{path}_2)$ indicates that search results must satisfy the conditions specified by both path_1 and path_2, while $(\mathit{path}_1 \vee \mathit{path}_2)$ indicates that they must satisfy those in path_1 or path_2.

Example 3. Consider the interface I_{ex}:

$$\big((F_1, \{\mathsf{President}\}) \wedge (F_3, \{d_{sp}, d_g\})\big) \wedge \big((F_2, \{\mathsf{any}\})/(F_3, \{\mathsf{any}\})\big).$$

It could be visualises it in our system SemFacet as in Figure 1, left. The interface consists of three paths connected by $\wedge$-branching. In the first path, we select documents

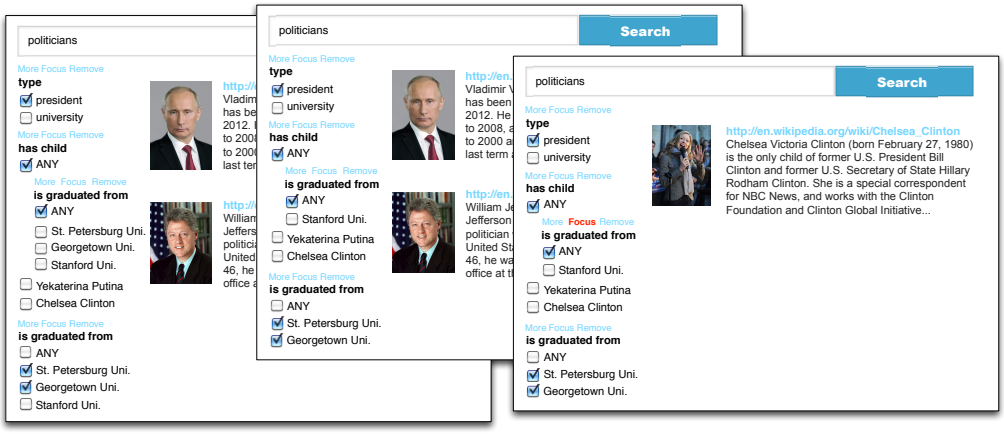

Fig. 1. Example interfaces implemented in SemFacet platform. Left: without minimisation; Center: after minimisation is applied; Right: minimised and refocussed.

categorised as 'president'. In the second path, we select graduates of St. Petersburg or Georgetown. In the third path, we first select individuals with a child who have some degree. Since paths are combined conjunctively, the constraints encoded in them apply simultaneously. Thus, we obtain the presidents who graduated from either St. Petersburg or Georgetown and who have a child graduated from some Uni. □

We formally specify the *semantics* of the queries encoded by the selected values in an interface in terms of first-order logic, see [30] for details. Intuitively, Our semantics assigns to each interface a PEQ with one free variable, which retrieves a set of documents. For each facet F we have that no restriction is imposed by F if no facet 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 $(path_1 \circ path_2)$ with $\circ \in \{\wedge, \vee\}$ is interpreted in the natural way by constructing the conjunction (disjunction) of the queries for each $path_i$; furthermore, if for some $path_i$ we have 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 subexpression. A first order formula φ is a *faceted query* if there exists a faceted interface I such that φ is are identical modulo renaming of variables to the semantics of I

Example 4. Our example interface $I_{\mathtt{ex}}$ encodes the positive existential query $Q_{\mathtt{ex}}(x)$:

$$\mathsf{President}(x) \wedge \left(\exists y_1 \left(\mathsf{grad}(x, y_1) \wedge y_1 \approx d_{sp}\right) \vee \exists y_2 \left(\mathsf{grad}(x, y_2) \wedge y_2 \approx d_g\right)\right) \\ \wedge \exists z \left(\mathsf{child}(x, z) \wedge \mathsf{grad}(z, w)\right).$$

If we consider only facts, the answer is Bill, since we do not know whether Yekaterina graduated from some university. If we also consider Rule (1), then both Vladimir and Bill are answers, since it says that one of the Vladimir's children has a degree. □

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.

Minimisation of Faceted 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. In [30] we provide interface minimisation algorithms. In Figure 1, center, we present a minimised version of the interface in Figure 1, left.

Faceted Interfaces with Refocussing. The interface in Example 3 finds two presidents. If we want to know who their children who guaranteed that these presidents returned by the query, (i.e., see Chelsea Clinton as an answer), we must provide *refocussing* (or *pivoting*) functionality [25, 26] that allows to change the output variable of faceted queries. In [30] we provide and extension of faceted interfaces to support refocussing. In Figure 1, right, we have a screenshot of SemFacet where the refocusing is set to the children and Chelsea is returned by the system. Note that Yekaterina is not in the answer set since we do not know whether she is the child of Vladimir with a degree.

3.2 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 affecting efficiency of such systems. As discussed above, faceted queries are monadic PEQs resulting from the selection of facet values in an interface. By standard results for relational databases, PEQ evaluation is an NP-hard problem, even for CQs over datasets. Our main result is that, in contrast to PEQs (and even CQs), faceted query evaluation over datasets is tractable, feasible in polynomial time; furthermore, the problem remains tractable in most cases if we consider ontologies belonging to the OWL 2 profiles. Our tractability results concern *combined complexity*, which takes into account the size of the entire input: ontological rules, RDF data and queries.

Active Domain Semantics. In practice, queries over ontology-enhanced RDF data are typically represented in SPARQL 1.1 and executed using off-the-shelf reasoning engines with SPARQL 1.1 support. The specification of SPARQL 1.1 under entailment regimes [36] 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 by first computing the dataset $\mathcal{D}$ of all facts entailed by $\mathcal{O}$ and then answers Q w.r.t. the dataset $\mathcal{D}$. Fact entailment is tractable for all the OWL 2 profiles; thus, by committing to the active domain semantics of SPARQL 1.1 we can achieve tractability without emasculating the ontology language.

Theorem 1. *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.*

Classical Semantics. Classical and active domain semantics coincide if we restrict ourselves to Datalog ontologies. Thus, the above result holds for query answering under classical semantics if the input ontology is Datalog. An immediate consequence is that our results in Theorem 1 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, we exploit techniques developed for the *combined approach* to query answering [37, 38]. These techniques are currently applicable to so-called *guarded* EL ontologies. The idea is to rewrite (EL) rules into Datalog by Skolemising existential variables. Although this transformation strengthens the ontology, it preserves the entailment of facts [37]. We showed [30] that the evaluation of faceted queries is also preserved. Thus, we conclude tractability of faceted query evaluation for EL. In contrast, we can show that faceted query evaluation is NP-complete for QL under classical semantics. The following theorem summarises our results.

Theorem 2. *Faceted query evaluation under classical semantics is (i)* PTIME*-complete for RL and guarded EL ontologies; and (ii)* NP*-complete for QL ontologies.*

In [30] we showed that theorems above hold for faceted queries with refocussing.

4 Interface Generation and 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. We propose to generate and update interfaces by 'traversing' the (explicit and implicit) information in $\mathcal{O}$. Our approach is based on a general principle: each element of the initial interface (resp. each change in an interface as a response of an action) must be 'justified' by a suitable entailment in $\mathcal{O}$. In this way, by exploring the ontology, we can 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. In [30] we resolve this ambiguities and devise fully deterministic algorithms for a restricted class of interfaces where conjunctive facets and disjunctive branching are disallowed. Example of such an interface is in Example 3 and in the screenshots in Figure 1.

The Ontology Facet Graph. We capture facets 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 $\mathcal{O}$. 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 $X = \mathsf{type}$, and (the constant) v is entailed to be an instance of (the unary predicate) w.

Definition 3. *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*, where the latter labels only edges from a constant to a unary*

predicate. 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 $c \xrightarrow{\mathsf{type}} A$, then $\alpha_e = A(c)$.

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 needed for a given application.

Example 5. A facet graph for the ontology in Example 1 may contain nodes for d_{bc} and d_{cc}, as well as for predicates such as President and Univ. Example edges are: a child-edge linking d_{bc} to d_{cc}, which is justified by the fact $\mathsf{child}(d_{bc}, d_{cc})$; and a grad-edge from d_{vp} to Univ due to Rule (1) in Example 1 and the facts about d_{vp}. □

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 1.1 queries. In this way, reasoning tasks associated to faceted search are performed offline.

Proposition 1. *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 any faceted interface over an ontology $\mathcal{O}$ *conforms* to the facet graph of $\mathcal{O}$, in the sense that the presence of every facet and value in the interface is supported by edges of the graph. In this way, we can ensure that interfaces mimic the structure of (and implicit information in) the ontology and, hence, that the interface does not contain irrelevant (combinations of) facets. Since a given facet or facet value can occur in many different places in an interface, we need a mechanism that allows us to refer to the elements of an interface in an unambiguous way. We refer to [30] for a formal definition of conformance. In [30] we also provide algorithms that by relying on facet graphs allow to create faceted interfaces (conformant to the graph), e.g., from sets of keywords, and to update such interfaces as a reaction on users actions, i.e., 'ticking' and 'unticking' operations on facet values.

5 SemFacet Platform

We have developed a faceted search platform SemFacet 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.

We bundled the platform in a prototypical system which powers faceted search over a fragment of Yago RDF data [33] enriched with DBPedia abstracts and OWL 2 RL Our system is available for downloading and installation, as well as a Web service [31]. In the remainder of this section, we will describe the system's workflow and architecture and present the results of SemFacet evaluation over both Yago and synthetic data.

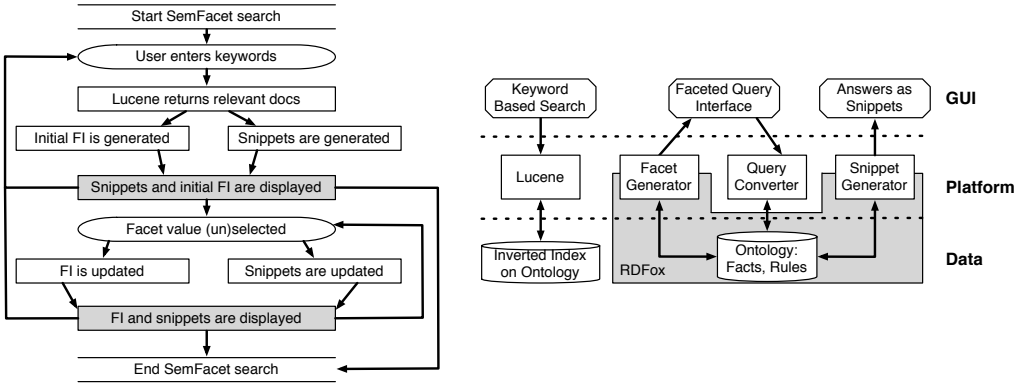

Fig. 2. Left: workflow of SemFacet, Right: architecture of SemFacet

5.1 System Architecture and Implementation

If Figure 2, left, we present the workflow diagram for SemFacet. The steps relevant to users' activity are depicted in ovals, and those relevant to SemFacet's activity are depicted as boxes. Grey boxes represent front-end activity and the remaining ones represent back-end tasks. Users start their interaction with SemFacet by entering a set of keywords, and the system returns as initial answers those documents that are relevant to the keywords; e.g., in Yago we considered as relevant those document URIs whose abstract contains at least one of the keywords. SemFacet displays the answers as snippets, combining the relevant URIs with the corresponding images and abstracts. The initial faceted interface is generated from the relevant URIs. Users can then perform faceted navigation by applying actions on the interface. SemFacet responds to each action by first updating the interface and then recomputing the query answers. Observe that in Figure 1 we nest facets to visualise the / operator and users can perform refocusing by clicking on the button 'focus' attached to facets.

The architecture of SemFacet is in Figure 2, right, where the components are arranged in three layers. SemFacet relies on RDFox [39][2] for storing RDF triples, and computing answers for SPARQL 1.1 queries. SemFacet converts faceted interfaces in SPARQL 1.1 queries and executes them using RDFox. Note that RDFox supports only conjunctive queries; thus, to answer faceted queries, we extended its query module with a support of `UNION`. To support keyword search, we load DBpedia abstracts in Lucene (`lucene.apache.org/`). Note that the implementation of SemFacet allows to substitute both Lucene an RDFox with any other software that provide the same functionality.

5.2 Experiments

We evaluated only the back-end of SemFacet (c.f. Figure 2), left. We excluded front-end activities since they highly depend on the client machine and network connection. In our experiments we evaluated the runtime performance to compute answer URIs and their associated snippet, as well as to update the interface. Our system presents ten snippets at time, thus, snippet generation time is negligible. Since we use Lucene as a black box, Lucene-related experiments are out of the scope of this paper.

Experimental Setup To evaluate the runtime performance of SemFacet, we used a MacBook Pro laptop with OS X 10.8.5, 2.4 GHz Intel Core i5 processor, and 8GB 1333

[2] RDFox is a parallel in-memory RDF triple store; it implements reasoning for OWL 2 RL; it materialises all implicit data via forward chaining and evaluates CQs over the materialisation.

# of queries	Avg time
10	0.0001 s
100	0.0181 s
1000	0.1449 s
10,000	1.1676 s
100,000	30.8467 s

(a) RDFox query time performance

# of facets	Avg time
1	0.0015 s
10	0.0122 s
100	0.0537 s
1000	0.3724 s
10,000	3.2964 s

(b) Interface construction run time

# of values	Avg time
1	0.005 s
10	0.0263 s
100	0.1054 s
1000	0.4573 s
10,000	3.3762 s

(c) Interface construction run time

# of values	Avg time
10	0.0027 s
100	0.0317 s
1000	0.1869 s
10,000	1.3101 s
100,000	22.6759 s

(d) Facet value hiding time

# of answers	# of facets	total # of values	Avg time
10	10	90	0.4918 s
100	15	727	0.5581 s
1000	21	6538	0.8616 s
10,000	28	51317	3.5472 s

(e) SemFacet time on the real data: Yago and DBpedia

Table 1. Experiment results for SemFacet

MHz DDR3 memory. The laptop runs the Apache Tomcat 7.0, with 4 GB of allocated memory. Each experiment presented in the following section was executed over 10 different runs to avoid cashing of objects. Each run was repeated 10 times in a FOR-loop to obtain a bigger sample size. Therefore, each experiment, having different parameters, was executed 100 times it total. We report average and median of running time performance results for each experiment.

Generation of Initial Interface SemFacet computes the initial interface by *(i)* gathering relevant facet names and values, and *(ii)* combining these in an interface.To perform Step (i), SemFacet sends to RDFox one atomic query for each relevant URI. Thus, to evaluate Step (i), we sent from 10 to 100,000 atomic queries to RDFox and measured response time. For this, we generated atomic queries and synthetic data containing exactly one answer per query Note that due to the RDFox indexing, evaluation time of atomic queries is constant regardless data size. The results are presented in Table 1a. Times are promising for up to 1,000 atomic queries (0.14 s), which corresponds to a keyword search that returns 1000 URIs; for $10,000$ queries, average runtime increases to $1.16s$ and for $100,000$ queries to $30.8s$. Thus, one may need to restrict the number of URIs returned by Lucene to guarantee reasonable response time for interface construction; in our online version of SemFacet, the output of Lucene is restricted to 10,000 URIs.

To evaluate Step (ii), we conducted two sets of experiments. In the first experiment we fixed the number of facets to 10 and increased the number of values in each facet from 1 to 10,000. In the second, we fixed the number of values per facet to 10 and increased the number of facets from 1 to 10,000. Results are presented in Tables 1b and 1c. Generation times for up to 1000 facets having 10 values each is promising (0.37 s). Generation times for 10,000 facets having 10 values each is increased to 3.3s on average. Moreover, the interface construction time is similar for n facets with m values each or for m facets with n values each. Note that the time in Tables 1b and 1c grows linearly.

Faceted Interface Update The experiments required to evaluate update of faceted interfaces are the same as those described for the generation of the initial interface [30]. Hence, we focus here on interface minimisation. We have evaluated our minimisation algorithm by generating faceted interfaces of growing sizes from 10 to 100,000 total facet values. Since RDFox performs reasoning by first materialising implicit data (a step performed only once) and then answering queries over the materialisation, we did not use an ontology in our experiments, and relied on synthetic RDF data. Results are

presented in Table 1d: the time to minimise 1000 values is promising (0.18s), and grows linearly upto 100,000 values (22.6s).

Experiments with Real Data Previous experiments were conducted to evaluate particular components of SemFacet in isolation. To evaluate the overall performance, we tested the system's running time on a fragment of Yago data enriched with DBpedia (`dbpedia.org/`) abstracts and OWL 2 RL axioms (around 5 million RDF triples in total). To mimic a real world scenario, we selected keywords that return from 10 to 10,000 answers. We measured the number of facets, the total number of values, and the response time of the system. The results of the experiments are presented in Table 1d: the time to obtain 1000 answers and to generate the corresponding interface is promising (0.86 s), and grows to 3.5 s for 10,000 answers. About one third of this time is spent by Lucene, one third by RDFox, and one third by the interface construction component of SemFacet.

6 Conclusion and Future Work

We presented a solid theoretical foundations for faceted search in the context of RDF and OWL. We have analysed the expressive power and complexity of queries stemming from faceted interfaces, and developed practical faceted navigation algorithms. Our results suggest many interesting problems for future work. In particular, we will explore extensions of our navigation algorithms beyond simple interfaces, as well as the design of more scalable interface minimisation algorithms. Concerning system design, substantial work is needed to improve GUI design, in particular with respect to advanced features such as refocusing. Finally, we are planning to conduct further experiments with knowledge bases other than Yago and perform user studies.

7 References

[1] E. Kharlamov, M. Giese, E. Jiménez-Ruiz, M. G. Skjæveland, A. Soylu, et al. Optique 1.0: Semantic Access to Big Data: The Case of Norwegian Petroleum Directorate's FactPages. In: *ISWC (Posters & Demos)*. 2013.

[2] E. Kharlamov, E. Jiménez-Ruiz, D. Zheleznyakov, D. Bilidas, M. Giese, et al. Optique: Towards OBDA Systems for Industry. In: *ESWC (Satellite Events)*. 2013.

[3] E. Kharlamov, N. Solomakhina, O. Ozcep, D. Zheleznyakov, T. Hubauer, et al. How Semantic Technologies can Enhance Data Access at Siemens Energy. In: *ISWC*. 2014.

[4] E. Kaufmann and A. Bernstein. Evaluating the usability of natural language query languages and interfaces to Semantic Web knowledge bases. In: *J. Web Sem.* 8.4 (2010).

[5] A. Bernstein, E. Kaufmann, A. Göhring, and C. Kiefer. Querying Ontologies: A Controlled English Interface for End-Users. In: *ISWC*. 2005.

[6] E. Franconi, P. Guagliardo, M. Trevisan, and S. Tessaris. Quelo: an Ontology-Driven Query Interface. In: *DL*. 2011.

[7] U. Waltinger, D. Tecuci, M. Olteanu, V. Mocanu, and S. Sullivan. Natural Language Access to Enterprise Data. In: *AI Magazine* 35.1 (2014).

[8] *Protege*. `http://protege.stanford.edu`.

[9] A. Soylu, M. G. Skjæveland, M. Giese, I. Horrocks, E. Jiménez-Ruiz, et al. A Preliminary Approach on Ontology-Based Visual Query Formulation for Big Data. In: *MTSR*. 2013.

[10] 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.

[11] D. Beneventano, S. Bergamaschi, F. Guerra, and M. Vincini. The SEWASIE Network of Mediator Agents for Semantic Search. In: 13.12 (Dec. 1, 2007). `http://www.jucs.org/jucs_13_12/the_sewasie_network_of`.

[12] A. Fadhil and V. Haarslev. OntoVQL: A Graphical Query Language for OWL Ontologies. In: *DL*. 2007.
[13] *iSPARQL QBE*. http://dbpedia.org/isparql/.
[14] 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.
[15] A. Russell and P. R. Smart. NITELIGHT: A Graphical Editor for SPARQL Queries. In: *ISWC (Posters & Demos)*. 2008.
[16] 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.
[17] P. Fafalios and Y. Tzitzikas. X-ENS: Semantic Enrichment of Web Search Results at Real-Time. In: *SIGIR*. 2013.
[18] R. Hahn, C. Bizer, C. Sahnwaldt, C. Herta, S. Robinson, et al. Faceted Wikipedia Search. In: *BIS*. 2010.
[19] D. Tunkelang. *Faceted Search*. Morgan & Claypool Publishers, 2009.
[20] E. Hyvönen, S. Saarela, and K. Viljanen. Ontogator: Combining View- and Ontology-Based Search with Semantic Browsing. In: *XML Finland*. 2003.
[21] 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.
[22] P. Heim, J. Ziegler, and S. Lohmann. gFacet: A Browser for the Web of Data. In: *IMC-SSW*. Vol. 417. 2008.
[23] M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A Browser for Heterogeneous Semantic Web Repositories. In: *ISWC*. 2006.
[24] 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).
[25] G. Kobilarov and I. Dickinson. Humboldt: Exploring Linked Data. In: *LDOW*. 2008.
[26] D. F. Huynh and D. R. Karger. Parallax and Companion: Set-based Browsing for the Data Web. 2013.
[27] E. Oren, R. Delbru, and S. Decker. Extending Faceted Navigation for RDF Data. In: *ISWC*. 2006.
[28] S. Ferré and A. Hermann. Semantic Search: Reconciling Expressive Querying and Exploratory Search. In: *ISWC*. 2011.
[29] A. Wagner, G. Ladwig, and T. Tran. Browsing-oriented Semantic Faceted Search. In: *DEXA*. 2011.
[30] M. Arenas, B. C. Grau, E. Kharlamov, S. Marciuska, and D. Zheleznyakov. Faceted Search over Ontology-Enhanced RDF Data. In: *CIKM*. 2014.
[31] *SemFacet*. http://www.cs.ox.ac.uk/isg/tools/SemFacet/.
[32] B. C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, M. Arenas, and E. Jimenez-Ruiz. SemFacet: Semantic Faceted Search over Yago. In: *WWW Demo*. 2014.
[33] F. M. Suchanek, G. Kasneci, and G. Weikum. Yago: A Core of Semantic Knowledge. In: *WWW*. 2007.
[34] B. Motik, B. Cuenca Grau, I. Horrocks, Z. Wu, A. Fokoue, and C. Lutz. OWL 2 Web Ontology Language Profiles. In: *W3C Recommendation* (2009).
[35] B. Motik, R. Shearer, and I. Horrocks. Hypertableau Reasoning for Description Logics. In: *J. Artif. Int. Res.* (2009).
[36] *W3C: SPARQL 1.1 Entailment Regimes*. www.w3.org/TR/sparql11-entailment/.
[37] G. Stefanoni, B. Motik, and I. Horrocks. Introducing Nominals to the Combined Query Answering Approaches for EL. In: *AAAI*. 2013.
[38] R. Kontchakov, C. Lutz, D. Toman, F. Wolter, and M. Zakharyaschev. The Combined Approach to Ontology-Based Data Access. In: *IJCAI*. 2011.
[39] *RDFox*. www.cs.ox.ac.uk/isg/tools/RDFox/.