The VLDB Journal (2023) 32:103–121
https://doi.org/10.1007/s00778-022-00735-3

REGULAR PAPER

Ontological databases with faceted queries

Tadeusz Pankowski[1]

Received: 6 December 2020 / Revised: 25 November 2021 / Accepted: 10 February 2022 / Published online: 15 March 2022

Abstract
The success of the use of ontology-based systems depends on efficient and user-friendly methods of formulating queries against the ontology. We propose a method to query a class of ontologies, called *facet ontologies* (*fac-ontologies*), using a faceted human-oriented approach. A fac-ontology has two important features: (a) a hierarchical view of it can be defined as a nested facet over this ontology and the view can be used as a faceted interface to create queries and to explore the ontology; (b) the ontology can be converted into an *ontological database*, the ABox of which is stored in a database, and the faceted queries are evaluated against this database. We show that the proposed faceted interface makes it possible to formulate queries that are semantically equivalent to $\mathcal{SROIQ}^{Fac}$, a limited version of the $\mathcal{SROIQ}$ description logic. The TBox of a fac-ontology is divided into a set of rules defining intensional predicates and a set of constraint rules to be satisfied by the database. We identify a class of so-called *reflexive weak cycles* in a set of constraint rules and propose a method to deal with them in the chase procedure. The considerations are illustrated with solutions implemented in the DAFO system (*data access based on faceted queries over ontologies*).

Keywords Ontological database · Faceted queries · Query building · Ontology query languages · Ontology-based data access

1 Introduction

In the last two decades, we have observed a steady increase in the number of applications based on ontology-oriented technologies. The reason is that ontologies provide a well-formulated and precise knowledge specification of the conceptualization of the application domain, and enrich query answering with intensional knowledge not explicitly captured by the extensional part of the ontology

It is known that classic reasoning problems in ontologies can be reduced to query answering problems [9]. Moreover, in many data-intensive applications, these inference capabilities are dominated by the need to respond to queries. To meet these needs, the extensional part of the ontology is often stored using robust and mature database technologies. Thus, we are witnessing the synergistic combination of ontologies and databases, which results in developing modern database solutions such as ontology-based data access (OBDA) [18,59,75], ontology-enhanced databases [7], ontological databases [37], and virtual knowledge graphs (VKG) [76].

✉ Tadeusz Pankowski
tadeusz.pankowski@put.poznan.pl

1 Institute of Computing Science, Poznań University of Technology, Poznan, Poland

Query and exploration tools for human-centered interaction remain an important issue in ontology-based access to data. To make a query interface easier to use, we propose a new concept of a *faceted view* of the ontology in a form of a hierarchical *nested facet* (as an extension of the "flat" facet investigated in [7]). The faceted view of an ontology depends on the intended query, whose template is provided by the user. A relevant part of the ontology, in the form of a spanning tree covering the query template, is produced in the response as a faceted view. A *faceted interface* is a faceted view equipped with a set of operations allowing for creating queries, and for extending the faceted view through ontology exploration. We propose such a set of operations that allows to create queries with the expressive power equivalent to (a slightly limited) $\mathcal{SROIQ}$ [44,48,61].

We assume that there are two sets of rules in a faceted ontology (*fac-ontology*: (1) $\mathcal{V}$ is a set of rules defining intensional predicates (views). Intensional predicates enrich the vocabulary and are used to facilitate query formulation. (2) $\mathcal{C}$ is a set of constraint rules which are expected to be satisfied by the ABox $\mathcal{A}$, i.e., they are materialized in the ABox by means of the chase procedure. We investigate the termination of the chase for a new class of weak cycles in $\mathcal{C}$, which we call *reflexive weak cycles*, and show that a chase terminates in the presence of reflexive weak cycles giving a solution (but not the universal solution). We do not consider rewriting rules because we assume that after the chase with respect to constraint rules, the ABox obeys all constraint rules. The ABox is stored in a relational database.

The presented approach has been implemented and verified in the DAFO system [24,55,57].

1.1 Contribution

The novelties in this paper are as follows:

1. Defining so-called *faceted ontologies* (fac-ontologies) and a concept of the *faceted view* over them. We use the faceted view to propose a *faceted interface* to create *faceted queries* over fac-ontologies. We show, that the faceted interface allows to formulate faceted queries equivalent (with some limitations) to queries (class expressions) in $\mathcal{SROIQ}$. Then, the query answering is LogSpace in the size of the ABox.
2. Identifying a new class of weak cycles, called *reflexive weak cycles* in the dependency graph of a class $\mathcal{C}$ of constraint rules in a fac-ontology. The proposed modification of the chase algorithm produces a solution, although the universal solution does not exist.

1.2 Outline

The paper is organized as follows. In Sect. 2, we define an ontology used as a running example, and formulate the motivations underlying the research. The notions of fac-ontologies and ontological databases are introduced in Sect. 3. In Sect. 4, we define the concept of a nested facet, which is next used to define faceted views, and faceted interfaces over ontological databases. Faceted queries are defined and investigated in Sect. 5. We show the relationship between faceted queries and queries (concept expressions) in the $\mathcal{SROIQ}$ description logic. Some subclasses of faceted queries and their graphical forms created by the faceted interface are presented in Sect. 6. In Sect 7, we discuss creation of an ontological database for a given fac-ontology, in particular we investigate the problem of the termination of the chase procedure in the presence of reflexive weak cycles. Related work and novelties of the paper are deeply analyzed in Sect 8. Section 9 summarizes and concludes the paper.

2 Running example and motivations

2.1 A sample ontology BibOn

As a running example, we consider an bibliographic ontology BibOn with the schema graph in Fig. 1. A schema graph is a directed graph with nodes labeled by classes, and edges labeled by properties. Unlabeled edges (with triangular arrows) denote subsumption relations between classes and between properties. Classes are *unary predicates*, and properties are *binary predicates*. Both classes and properties are divided into *extensional* and *intensional* ones. Extensional predicates are materialized in the ABox, while intensional are views defined by rules. There are two rationale behind using intensional predicates.

1. *Simplification of the syntax of rules.* In most implemented systems, classes and properties are restricted to being atomic names [9]. This can be achieved by using intensional predicates (views). For example, the intensional predicate (class) *ACMConf* can be defined by the following *definition rule*:

$$Conference \sqcap \exists organizer.\{'ACM'\} \equiv ACMConf.$$

Then *ACMConf* is an atomic class name and its definition consists of extensional predicates $Conference$ and $organizer$ and a constant $'ACM'$. Similarly, the definition rule

$$Paper \sqcap \exists presentedAt.ACMConf \equiv ACMPaper$$

defines a subclass *ACMPaper*. Then *ACMPaper* is an intensional predicate that can be unfolded to an extensional expression. Then, for example, the subsumption $ACMPaper \sqsubseteq Paper$, abbreviates the following subsumption between class expressions:

$$Paper \sqcap \exists presentedAt.(Conference \sqcap \exists organizer.\{'ACM'\}) \sqsubseteq Paper.$$

2. *Enriching the vocabulary of the ontology.* Intensional predicates enrich the vocabulary of the ontology, thus facilitating the formulation of queries. They are, in fact, views defined over extensional predicates. During query rewriting, they are unfolded using their definitions.

In Fig. 1, extensional classes and properties are drawn with solid lines, while intensional with dashed lines.

Below, we list some rules in the BibOn using the standard description logic notation.

1. There are four *inheritance hierarchies* with uniquely defined extensional *top classes*: *Person*, *Paper*, *Proceedings*, and *Conferences*, respectively. They are pairwise disjoint, e.g.,
   - $Person \sqsubseteq \neg Paper.$

   The distinguished class *Data* is a class of standard data values (strings, numbers, etc.).

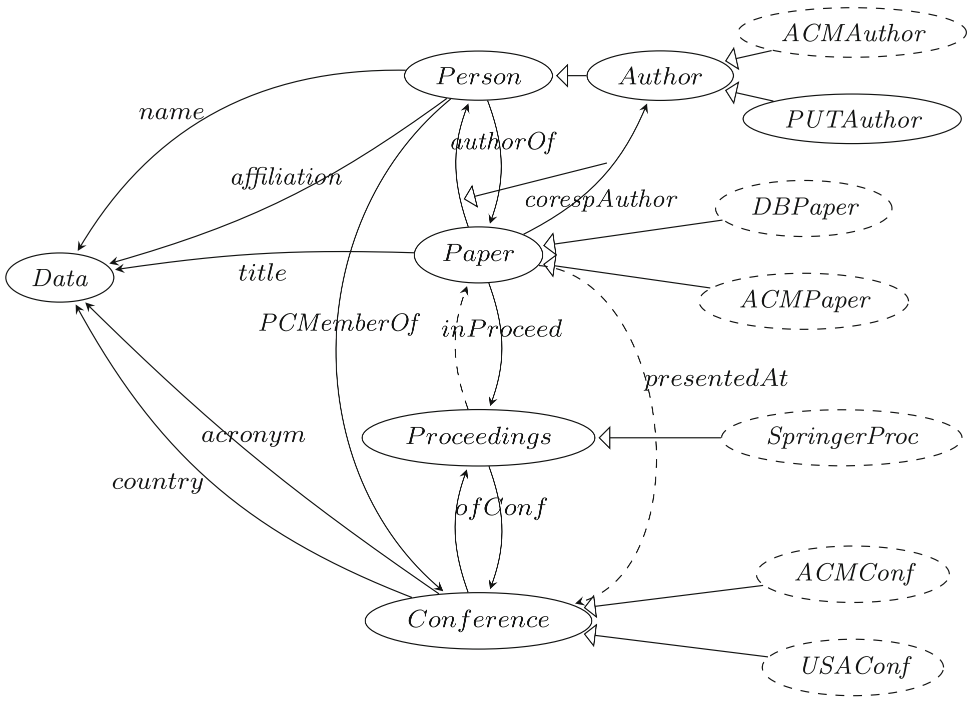


Fig. 1 A graph of the BibOn ontological schema

2. *Subsumption and equivalences between extensional classes and properties.*
   - Class subsumption: $Author \sqsubseteq Person$, $PUTAuthor \sqsubseteq Author$.
   - Class-driven specialization (definition of extensional predicate): $Person \sqcap \exists authorOf.Paper \equiv Author$.
   - Value-driven specialization (definition of extensional predicate): $Author \sqcap \exists affiliation.\{'PUT'\} \equiv PUTAuthor$.
   - Property subsumption: $correspAuthor \sqsubseteq writtenBy$.
3. *Definitions of intensional predicates*:
   - $Author \sqcap \exists authorOf.ACMPaper \equiv ACMAuthor$.
   - $Conference \sqcap \exists country.\{'USA'\} \equiv USAConf$.
   - $inProceed^{-} \equiv includesPaper$.
   - chains of properties: $inProceed \circ ofConf \equiv presentedAt$, $authorOf \circ presentedAt \equiv authorConf$, $authorConf \circ confPCMember \equiv authConfPCMember$.
4. *Domains and ranges of properties.* If P is a property, then $\exists P \sqsubseteq A$ specifies that the domain of P is subsumed by A. We denote by $dom(P)$ the *least upper bound* of the set of classes subsumming $\exists P$. Similarly, by $rng(P)$ we denote the least upper bound of classes subsuming $\exists P^{-}$, e.g.,
   - $dom(name) = Person$, $rng(name) = Data$,
   - $dom(correspAuthor) = Paper$,
   - $rng(correspAuthor) = Author$.
5. *Mandatory membership of classes (or totality of properties).* $A \sqsubseteq \exists P$ specifies that A has the *mandatory* membership in the domain of P, or that P is *total* on A. Similarly, $A \sqsubseteq \exists P^{-}$ says that A has the *mandatory* membership in the range of P, or that P^{-} is *total* on A, e.g.,
   - $Person \sqsubseteq \exists name$,
   - $Author \sqsubseteq \exists writtenBy^{-}$.
6. *Functionality of properties*:
   - *name* is a functional data property: (`funct` *name*),
   - $inProceed$ is a functional object property: (`funct` $inProceed$).

2.2 Motivation of the paper

In Fig. 2, we show a faceted query tree that formulates the request:

"Get persons who: (a) are authors of at least ten papers presented at ACM or IEEE conferences, and (b) are not affiliated at the 'PUT' ('Poznan University of Technology'), and (c) served PC members at conferences where they presented their papers."

In the faceted query tree in Fig. 2:

1. The root is labeled by the distinguished property *root*, we assume that *root* is reflexive universal property, i.e., $\forall x, y(root(x, y) \leftrightarrow x = y)$.

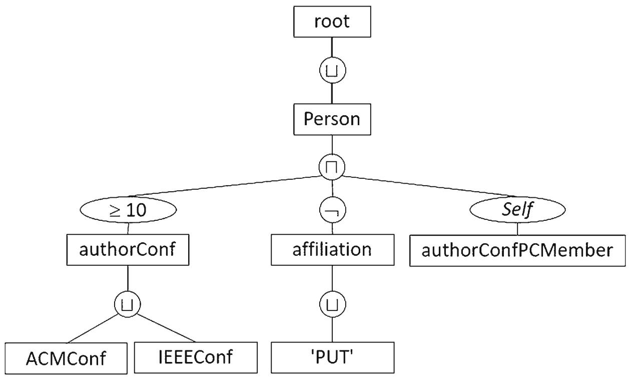


Fig. 2 A sample faceted query tree over the BibOn ontology

2. Every node (rectangle) is a class-node (labeled by a class), a property-node (labeled by a property) or a constant-node (labeled by a constant).
3. A node is labeled either by "$\sqcup$" or "$\sqcap$"—the label is drawn below the node, and the set of children is then either disjunctive or conjunctive.
4. Nodes are labeled by: a number restriction ($\geq$ 10), negation ($\neg$), or local reflexivity (*Self*)—these labels are drawn above the node.
5. The semantics of the query is defined by its translation to a DL $\mathcal{SROIQ}$ expression: $\exists root.(Person \sqcap ((\geq 10) authorConf.(ACMConf \sqcup IEEEConf) \sqcap \neg\exists affiliation.\{'PUT'\} \sqcap \exists authConfPCMember.Self))$. Note that $\exists root.\ Q \equiv Q$, so $\exists root$ can be omitted.

The tree-shaped structure of a faceted query also provides a faceted view of an ontology. In order to be viewed and queried in this way, an ontology must satisfy properties implied by the following assumptions:

1. All the nodes connected by $\sqcup$ or $\sqcap$ have a common parent node, and all of them are in the same inheritance hierarchy. The set of inheritance hierarchies is pairwise disjoint, and each hierarchy has a unique top class (the least upper bound). The top class is also used to compute the negation of a query (the complement with respect to the top class).
2. Every path of nodes in a faceted query tree is of the form: $(root, A_1, \ldots, P_n, A_n)$, $(root, A_1, \ldots, P_n)$, or $(root, A_1, \ldots, P_n, a_n)$, $n \geq 1$, where A, P, and a (with subscripts) are a class, a property, or a constant, respectively. This can be achieved if every class is defined as a specialization of its superclass.
3. The set of all predicates is divided into extensional (can occur in the ABox $\mathcal{A}$) and intensional predicates (defined by rules). The extensional predicates are materialized by means of a chase procedure in the ABox and in a database.

The running example and the above comments serve as a motivation for the following research problems presented in this paper.

1. *Faceted ontologies* (fac-ontologies). We identify fac-ontologies as a class for which the proposed faceted approach can be applied.
2. *Faceted-oriented query formulation*. We develop a method based on the faceted approach. We examine the expressive power of this approach and compare it with other faceted-oriented query systems.
3. *Creating a (faceted) ontological database for answering faceted queries*. We define and consider the so-called *reflexive weak cycles*, and propose a method of chasing the ABox in the presence of these cycles.

3 Ontological database

Ontologies are commonly considered as the best method to specify conceptualizations of application domains (conceptual models) [8]. In a database design, a conceptual model is presented as the entity–relationship (ER) diagram [22], expanded entity–relationship (EER) diagram [28] or an UML class diagram. A family of ontologies which can be used to a formal specification of such conceptual models is the *DL-Lite* family [8,17]. The importance and usefulness of the *DL-Lite* family is testified by the fact that it is the basis of the W3C OWL 2 QL standard [54]. In this paper, we define a class of ontologies called *faceted ontologies* (*fac-ontologies*). In general, from a methodological point of view, each fac-ontology has the following properties:

1. The terminological part of the fac-ontology is a formal specification of the conceptual schema of an application domain created by means of the EER model. In particular, we follow the idea of subclasses specification by means of the *specialization* abstraction [28].
2. Every finite part of the terminological part of the fac-ontology must be representable through a faceted interface (Sect. 4.3). In particular, in any class hierarchy, each subclass is a specialization of the top class of this hierarchy.

3.1 Faceted ontologies and ontological database

In this section, we define an *ontological database*. We denote by $\mathsf{UP} = \mathsf{UP}_E \cup \mathsf{UP}_I$ an infinite set of *unary predicates* (classes), where UP_E is a set of extensional, and UP_I a set of intensional classes. Analogously, we denote by $\mathsf{BP} = \mathsf{BP}_E \cup \mathsf{BP}_I$ an infinite set of *binary predicates* (properties) consisting of a set of extensional (BP_E) and intensional (BP_I) properties. Const denotes an infinite set of *constants* (individual names). We also assume that a set of *labeled nulls*, $\mathsf{LabNull} \subseteq \mathsf{Const}$, is a subset of constants, for which the UNA (*Unique Name Assumption*) does not hold [1]. In particular, two different labeled nulls can denote the same individual, i.e., $\mathsf{N}_1^{\mathcal{I}} = \mathsf{N}_2^{\mathcal{I}}$ for an interpretation $\mathcal{I}$. Regular constants, i.e., constants from $\mathsf{Const} \setminus \mathsf{LabNull}$, obey UNA. It means that two different regular constants always denote different individuals.

Data is a distinguished class of *data values* (strings, numbers, etc.). The other classes are *object classes* and their instances are *objects*. Both data values and objects are represented by constants. Every property with the range in the class *Data* is a *data property*; otherwise, it is an *object property*.

We define now the syntax for rules.

Definition 1 Let A, P, and a be, respectively, a class, a property, and a constant (individual name). Let:

$$C ::= A \mid \exists R \mid \exists R.\{a\} \mid \exists R.C \mid C \sqcap C,$$
$$R ::= P \mid P^-,$$
$$S ::= R \mid S \circ R.$$

Then

1. *Definition rules* are rules with the syntax:

   $$\mathcal{L}_{\mathcal{V}} ::= C \equiv A \mid S \equiv P \mid dom(P) = A \mid rng(P) = A.$$

2. *Constraint rules* are rules built from extensional predicates and conforming to the syntax:

   $$\mathcal{L}_{\mathcal{C}} ::= C \sqsubseteq C \mid S \sqsubseteq S \mid A \sqsubseteq \neg A \mid R \sqsubseteq \neg R \mid (\mathsf{funct}\ S).$$

Definition 2 A set Class of classes is an *inheritance hierarchy* if it is a *finite bounded complete partial order*,

$$\mathcal{H} = (\top^{\mathcal{H}}, \mathsf{Class}, \sqsubseteq),$$

where $\top^{\mathcal{H}} \in \mathsf{Class}$ is the *least upper bound* (the *top class*) in $\mathcal{H}$.

We will also denote by $\top^A$ the top class $\top^{\mathcal{H}}$ of an inheritance hierarchy to which belongs a class A.

Definition 3 A class A is a *specialization* of a class A_0 in a set of rules if the rule $A_0 \sqcap C \equiv A$ is derivable in this set, and the syntax of C is given in Definition 1.

Definition 4 An *fac-ontology* with the signature $Sig(\mathcal{O}) \subseteq \mathsf{UP} \cup \mathsf{BP} \cup \mathsf{Const}$ is a triple $\mathcal{O} = (\mathcal{V}, \mathcal{C}, \mathcal{A})$ such that:

1. $\mathcal{V}$, and $\mathcal{C}$ are sets of rules with the syntax defined by $\mathcal{L}_{\mathcal{V}}$, and $\mathcal{L}_{\mathcal{C}}$, respectively, and $\mathcal{A}$ is a set of facts of the form $A(a)$, and $P(a_1, a_2)$, where A and P are extensional predicates, and a, a_1, a_2 are constants.
2. The set UP of classes form a set of pairwise disjoint inheritance hierarchies with extensional top classes.
3. Every class A is either the top class in its inheritance hierarchy or is a specialization of $\top^A$, denoted as $\top^A \sqcap C \equiv A$.

Definition 5 An *ontological database* is a fac-ontology $\mathcal{O} = (\mathcal{V}, \mathcal{C}, \mathcal{A})$ such that $\mathcal{A}$ satisfies all rules in $\mathcal{C}$, i.e., $\mathcal{A} \models \mathcal{C}$. We denote this as $\mathcal{O}_{db} = (\mathcal{V}, \mathcal{C}, \mathcal{A}_{db})$.

Example 1 In the BibOn fac-ontology:

1. $\mathcal{V}$ can contain: $Paper \sqcap \exists presentedAt.ACMConf \equiv ACMPaper$, $Conference \sqcap \exists country.\{\text{'}USA'\} \equiv USAConf$, $inProceed^- \equiv includesPaper$, $inProceed \circ ofConf \equiv presentedAt$.
2. $\mathcal{C}$ can contain: $Person \sqcap \exists authorOf.Paper \equiv Author$, $Author \sqcap \exists affiliation.\{'PUT'\} \equiv PUTAuthor$, $Author \sqsubseteq \exists authorOf.Paper$, $Paper \sqsubseteq \exists writtenBy.Author$, $PUTAuthor \sqsubseteq Author$, $authorOf \sqsubseteq writtenBy^-$, $Person \sqsubseteq \neg Paper$, $(\mathsf{funct}\ ofConf)$.

3.2 $\mathcal{SROIQ}^{Fac}$—a subset of $\mathcal{SROIQ}$

We will consider $\mathcal{SROIQ}^{Fac}$ as a query language, which is a subset of $\mathcal{SROIQ}$ [44,46,48,61]. The syntax of $\mathcal{SROIQ}^{Fac}$ is defined by the grammar

$$\begin{aligned} q &::= A \mid \{a\} \mid q \sqcap q \mid q \sqcup q \mid \neg q \\ &\quad \mid \exists R \mid \exists R.q \mid (\geq k)R \mid (\geq k)R.q \mid \exists R.Self \\ R &::= P \mid P^-, \end{aligned} \tag{1}$$

where A is a class, P is a property, a is a constant, and k is an integer, $k \geq 1$.

We will use $\mathcal{SROIQ}^{Fac}$ as a reference language for faceted queries and assume that it satisfies the following restrictions:

1. A nominal $\{a\}$ can occur only in extensional restrictions, i.e., in $\exists R.\{a\}$.
2. Every query q in $\mathcal{SROIQ}^{Fac}$ has a uniquely determined type, $type(q)$, where:

   $$\begin{aligned} &type(A) = \top^A,\ type(\{a\}) = Data, \\ &type(q_1 \sqcap q_2) = type(q_1) = type(q_2), \\ &type(q_1 \sqcup q_2) = type(q_1) = type(q_2), \\ &type(\neg q) = type(q), \\ &type(\exists R) = type(\exists R.q) = type((\geq k)R) = \\ &= type((\geq k)R.q) = type(\exists R.Self) = type(dom(R)). \end{aligned}$$

Formally, the semantics of $\mathcal{SROIQ}^{Fac}$ is defined in terms of an interpretation $\mathcal{I}$ consisting of a non-empty set $\Delta^{\mathcal{I}}$ (the domain of the interpretation) and an interpretation function

Table 1 Syntax and semantics of $\mathcal{SROIQ}^{Fac}$

	Syntax	Semantics
Property inversion	P^-	$\{(x, y) \mid (y, x) \in P^\mathcal{I}\}$
Nominal	$\{a\}$	$\{a^\mathcal{I}\}$
Conjunction	$q_1 \sqcap q_2$	$q_1^\mathcal{I} \cap q_2^\mathcal{I}$
Disjunction	$q_1 \sqcup q_2$	$q_1^\mathcal{I} \cup q_2^\mathcal{I}$
Existential	$\exists R$	$\{x \in \Delta^\mathcal{I} \mid \exists y (x, y) \in R^\mathcal{I}\}$
Restriction	$\exists R.q$	$\{x \in \Delta^\mathcal{I} \mid \exists y (y \in q^\mathcal{I} \wedge (x, y) \in R^\mathcal{I})\}$
$(\geq k)$ restriction	$(\geq k)R$	$\{x \in \Delta^\mathcal{I} \mid \#\{y \in \Delta^\mathcal{I} \wedge (x, y) \in R^\mathcal{I}\} \geq k\}$
	$(\geq k)R.q$	$\{x \in \Delta^\mathcal{I} \mid \#\{y \in q^\mathcal{I} \wedge (x, y) \in R^\mathcal{I}\} \geq k\}$
Local reflexivity	$\exists R.Self$	$\{x \mid (x, x) \in R^\mathcal{I}\}$
Negation	$\neg q$	$(type(q))^\mathcal{I} \setminus q^\mathcal{I}$

$\cdot^\mathcal{I}$, which assigns: to every atomic class A a set $A^\mathcal{I} \subseteq \Delta^\mathcal{I}$, to every atomic property P a binary relation $P^\mathcal{I} \subseteq \Delta^\mathcal{I} \times \Delta^\mathcal{I}$, and to every data name a an element $a^\mathcal{I} \in Data^\mathcal{I}$. Table 1 shows how to obtain the semantics of each compound expression from the semantics of its parts [9,48].

Note that the semantics of the negation of q is defined relatively to the $type(q)$, i.e., to the top class (the "local universal class") of the inheritance hierarchy containing the class of answers to q.

An interpretation $\mathcal{I}$ satisfies the subsumptions $C_1 \sqsubseteq C_2$ and $S_1 \sqsubseteq S_2$ if $C_1^\mathcal{I} \subseteq C_2^\mathcal{I}$ and $S_1^\mathcal{I} \subseteq S_2^\mathcal{I}$, respectively. $\mathcal{I}$ satisfies $A_1 \sqsubseteq \neg A_2$ and $R_1 \sqsubseteq \neg R_2$ if $A_1^\mathcal{I} \cap A_2^\mathcal{I} = \emptyset$, and $R_1^\mathcal{I} \cap R_2^\mathcal{I} = \emptyset$, respectively, and satisfies (funct S) if $S^\mathcal{I}$ is a partial function. We say that $\mathcal{I}$ is a *model* of a TBox or an ABox if it satisfies all subsumptions and facts in it. An ABox $\mathcal{A}$ is consistent with $\mathcal{C}$ if $\mathcal{A}$ and $\mathcal{C}$ have a common model. If $\mathcal{I}$ is a model of $\mathcal{O}$, then we denote this as $\mathcal{I} \models \mathcal{O}$, or $\mathcal{I} \models \mathcal{A} \cup \mathcal{V} \cup \mathcal{C}$.

3.3 Query answering in ontological databases

Let q by a query in $\mathcal{SROIQ}^{Fac}$. We denote by $q(x)$ a first-order form of the query q, which can be obtained using rules proposed in [61]. A *certain answer* to $q(x)$ over a fac-ontology $\mathcal{O} = (\mathcal{V}, \mathcal{C}, \mathcal{A})$ is a constant a for each $q(a)$ is satisfied for all models of $\mathcal{O}$, denoted $\mathcal{A} \cup \mathcal{V} \cup \mathcal{C} \models q(a)$, or shortly $\mathcal{O} \models q(a)$. The set of all certain answers to $q(x)$ over $\mathcal{O}$ is denoted as $q^{(\mathcal{O})}$, or $q^{(\mathcal{A} \cup \mathcal{V} \cup \mathcal{C})}$.

If $\mathcal{O}_{db} = (\mathcal{V}, \mathcal{C}, \mathcal{A}_{db})$ is an ontological database, then $\mathcal{C}$ is satisfied in $\mathcal{A}_{db}$, and is immaterial in query answering, i.e., the equality holds:

$$q^{(\mathcal{O}_{db})} = q^{(\mathcal{A}_{db} \cup \mathcal{V})}.$$

To eliminate intensional predicates in q, we unfold rules in $\mathcal{V}$. Then every intensional predicate is replaced by its extensional definition. We obtain a query $q_{[\mathcal{V}]}$, in which only extensional predicates appear, and the equality holds:

$$q^{(\mathcal{O}_{db})} = q_{[\mathcal{V}]}^{(\mathcal{A}_{db})}.$$

In DAFO, a mapping $\mathcal{M}$ is used to map $\mathcal{O}_{db}$ into a relational database $\mathcal{D} = (Sch, D)$, where the instance D is created from $\mathcal{A}_{db}$, i.e., $\mathcal{M}(\mathcal{A}_{db}) = D$, and $\mathcal{M}$ is a set of dependencies of the form:

$$\forall \mathbf{x} (\alpha(\mathbf{x}) \rightarrow \exists \mathbf{y} \mathtt{R}(\mathbf{x}, \mathbf{y})),$$

where α ranges over unary and binary atoms, and $\mathtt{R}$ is an n-ary relation name in Sch, $n \geq 1$. Additionally, we assume that D only has data "exported" from $\mathcal{A}_{db}$, and each labeled null is mapped to `NULL`.

The mapping $\mathcal{M}$ is used in the translation of $q_{[\mathcal{V}]}$ into a SQL query $q_{[\mathcal{V},\mathcal{M}]}$ over the database D. As a result, the set of answers to q over $\mathcal{O}_{db}$ coincides to the set of answers to $q_{[\mathcal{V},\mathcal{M}]}$ over the database, i.e.,

$$q^{(\mathcal{O}_{db})} = q_{[\mathcal{V},\mathcal{M}]}^{(D)}.$$

4 Faceted views and faceted interfaces over ontological databases

We now propose a method of query formulation against an ontological database $\mathcal{O}_{db} = (\mathcal{V}, \mathcal{C}, \mathcal{A}_{db})$. The method is based on the idea of *faceted search* [70] and *faceted queries* [7].

We start with a concept of the *nested facet* as a generalization of the ("flat") facet proposed in [7].

4.1 Nested facet

In [7], a *facet* is defined as a pair $(X, \Diamond\Gamma)$, with $\Diamond \in \{\sqcap, \sqcup\}$, Γ a non-empty set, where either: (a) $X =$ type and Γ is a set of classes, or (b) X is a binary predicate (a property) and Γ contains a distinguished symbol any and a set of individual names (constants) or a set of classes. A facet of the form $(X, \sqcap\Gamma)$ is conjunctive, and a facet of the form $(X, \sqcup\Gamma)$ is disjunctive.

We will extend the above notion of facets to *nested facets*, which have the form of a tree. In this tree-oriented notation, a facet $(X, \Diamond\Gamma)$ will be written as a tree $\Diamond X(\Gamma)$, where $\Diamond X$ is the labeled root, and Γ is a set of its children, interpreted as a conjunctive or disjunctive set depending on $\Diamond$. Note that a facet defined in [7] forms always a two level tree.

In commercial applications, faceted queries are usually limited to flat facets where the flat facet corresponds to possible values of one data property. A query consists then of several facets, each of which relates to a different data property (aspect) of the searched object (e.g., price, manufacture, size, in the case of mobile phones). We extend this approach by introducing object properties and subsumptions between classes and properties. This requires nested faceted queries. The nested form of faceted queries is also due to the fact that a faceted query is a (complex) Description Logic query, and its syntax tree is nested.

We define a nested facet as a finite multi-level tree with at least two levels.

Definition 6 Let A, P, and a, possibly with subscripts, be a class, a property and a constant, respectively. Let $root$ be a distinguished property not in BP, and $\Diamond \in \{\sqcap, \sqcup\}$. A *nested facet* f is an expression with the syntax defined by the grammar:

$$\begin{aligned} f &::= \Diamond root\{u_1, \ldots, u_k\}, \\ u &::= A \mid \Diamond A\{t_1, \ldots, t_l\}, \\ t &::= P \mid P\{a_1, \ldots, a_m\} \mid \Diamond P\{u_1, \ldots, u_n\}. \end{aligned}$$

where $k, l, m, n \geq 1$.

Every expression $\Diamond X\{Y_1, \ldots, Y_n\}$, $n \geq 0$, of the category f, u, and t is a tree with the root X and a set (possibly empty) of subtrees, which are children of X. The tree is labeled by $\Diamond$. By default, if $X \in \mathsf{BP} \cup \{root\}$, then $\Diamond = \sqcup$, otherwise $\Diamond = \sqcap$, i.e., the set of property-node children is disjunctive, while the set of class-node children is conjunctive. $P\{a_1, \ldots, a_n\}$ abbreviates the tree $\sqcup P\{\{a_1\}, \ldots, \{a_n\}\}$. The trees $A\{\}$ and $P\{\}$ are abbreviated by A and P, respectively.

Example 2 A nested facet corresponding to the sentence *"Persons who are authors of papers presented at ACM or IEEE conferences, and affiliated in PUT"*, is:

$$\begin{array}{l} \sqcup root\{ \\ \quad \sqcap Person\{ \\ \quad\quad \sqcup authorConf\{ACMConf, IEEEConf\}, \\ \quad\quad affiliation.\{'PUT'\} \\ \quad \} \\ \}, \end{array}$$

Its graphical form is shown in Fig. 3. Note that in the graphical representation, the logical connective is drawn below the labeled node.

Its semantics is given by the following query in $\mathcal{SROIQ}^{Fac}$ (a formal translation is given later on in Definition 10):

$$Person \sqcap (\exists authorConf.(ACMConf \sqcup IEEEConf) \sqcap \exists affiliation.\{'PUT'\}).$$

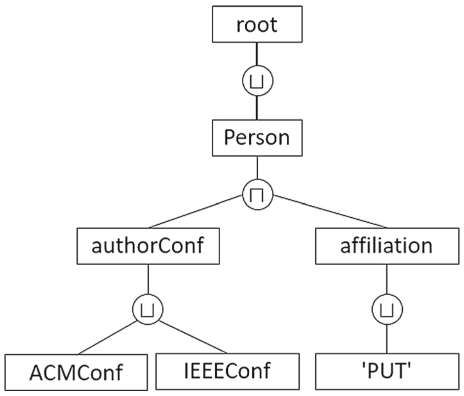


Fig. 3 A graphical form of the nested facet in Example 2

4.2 Faceted view

If a nested facet relates to a fac-ontology, then we call it a *faceted view* of this ontology. If u is a class-node of a nested facet (Definition 6), then we denote by $\rho(u)$ the root class of u, i.e., $\rho(\Diamond A\{t_1, \ldots, t_n\}) = A$, and similarly, by $\rho(t)$ we denote the root property of t.

Definition 7 A nested facet f is a *faceted view* over an fac-ontology $\mathcal{O} = (\mathcal{V}, \mathcal{C}, \mathcal{A})$ if the signature of f is in the signature of $\mathcal{O}$, and

1. If $\Diamond root\{u_1, \ldots, u_n\}$ is in f and $\{\rho(u_1), \ldots, \rho(u_n)\} = \{A_1, \ldots, A_n\}$, then all classes in $\{A_1, \ldots, A_n\}$ are in the same inheritance hierarchy, i.e., there is the unique least upper bound (top class) of them.
2. If $\Diamond A\{t_1, \ldots, t_n\}$ is in f and $\{\rho(t_1), \ldots, \rho(t_n)\} = \{P_1, \ldots, P_n\}$, then the domain of every property P_i is subsumed by A, i.e., $dom(P_i) \sqsubseteq A$, for $1 \leq i \leq n$.
3. If $\Diamond P\{u_1, \ldots, u_n\}$ is in f and $\{\rho(u_1), \ldots, \rho(u_n)\} = \{A_1, \ldots, A_n\}$, then every class A_i is subsumed by the range of P, i.e., $A_i \sqsubseteq rng(P)$, for $1 \leq i \leq n$.
4. If $P\{a_1, \ldots, a_n\}$ is in f, then $\exists x\ P(x, a_i)$ is satisfied in $\mathcal{O}$, i.e., $\mathcal{O} \models \exists x\ P(x, a_i)$, for $1 \leq i \leq n$.

In contrast to a structural graph (Fig. 1) that provides a *graph-oriented view* of the entire ontology, the aim of a *faceted view* is to provide a *tree-oriented view* (a *hierarchical view*) of a part of this ontology relevant for the intended query.

A faceted view over the ontology is the basis of faceted search, where the formulation of queries proceeds over a hierarchical view of this ontology [25,77].

4.3 Faceted interface

We show now how a faceted view over a fac-ontology can be used to built a *faceted interface* supporting query formulation. Intuitively, a faceted interface is a faceted view equipped with some operational features allowing to label, extend and narrow the view.

Definition 8 A *faceted interface* over a fac-ontology $\mathcal{O}$, with a signature $Sig(\mathcal{O}) = \mathsf{UP} \cup \mathsf{BP} \cup \mathsf{Const}$, is a labeled tree $FI = (r, N, E, \lambda, State)$ rooted in $r \in N$, such that:

1. N is a set on nodes, $E \subseteq N \times N$ is a set of edges defining a tree rooted in r, and λ is a node labeling function ($N_L \subseteq N$ is a set of leaves):
   - $\lambda : N \setminus N_L \to \{\sqcap, \sqcup\} \times (\{root\} \cup \mathsf{UP} \cup \mathsf{BP})$,
   - $\lambda : N_L \to \mathsf{UP} \cup \mathsf{BP} \cup \mathsf{Const}$,

   that assigns a pair $(\Diamond, name)$, $\Diamond \in \{\sqcap, \sqcup\}$, $name \in \{root\} \cup \mathsf{UP} \cup \mathsf{BP}$ to every non-leaf node, and a $name \in \mathsf{UP} \cup \mathsf{BP} \cup \mathsf{Const}$ to any leaf node.
2. λ defines a faceted view over $\mathcal{O}$, meaning that the result of the *depth-first-search* (DFS) serialization of FI is a faceted view over $\mathcal{O}$.
3. A state of each node $v \in N$ is defined by the quintuple function:

   $$State(v) = (Sel(v), AndOr(v), PosNeg(v), NumRestr(v), Self(v)),$$

   where
   - $Sel(v) \in \{\varepsilon, \mathtt{YES}\}$—indicates whether v is selected (checked) or not.
   - $AndOr(v) \in \{\sqcap, \sqcup\}$—indicates that the set of children of a non-leaf node v is conjunctive ($\sqcap$) or disjunctive ($\sqcup$).
   - $PosNeg(v) \in \{\varepsilon, \neg\}$—indicates whether the subtree rooted in v is negated (*excluded*) or not.
   - $NumRestr(v) \in \{\varepsilon, (\geq, k)\}$—indicates if a number restriction is bound to a property node v.
   - $Self(v) \in \{\varepsilon, Self\}$—indicates if a local reflexivity restriction is bound to an object property node v.

In Fig. 4, there is a faceted interface in the form implemented in the DAFO system. On the left-hand side, we see a sample initial faceted interface on which a user can operate using operations presented in the context menu shown on the right-hand side.

A faceted interface in DAFO is implemented as a labeled AND/OR tree such that:

1. Nodes are labeled by classes (*class nodes*), properties (*property nodes*), and constants (*constant nodes*). Constant nodes are visible only on demand.
2. There is a distinguished root, which is a property node labeled by the universal reflexive property *root*.

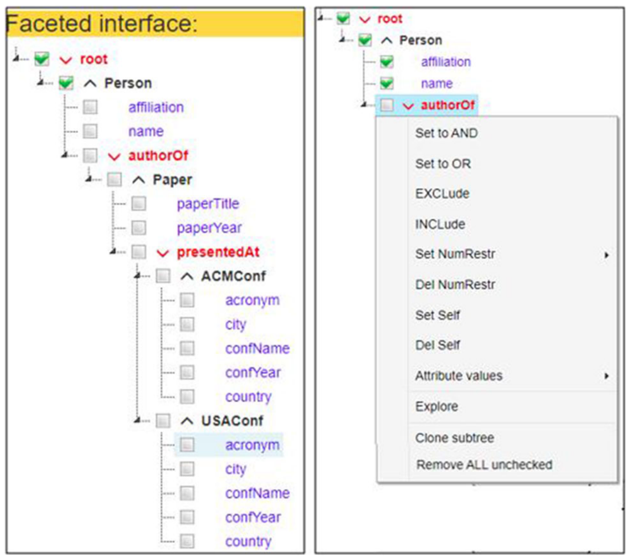


Fig. 4 A sample faceted interface in the DAFO system and the context menu containing operations on it

3. At the beginning, any class node has the black color and the AND ("∧") label meaning that the set of its children is interpreted as a *conjunctive set*, while each property node has the red color and the OR ("∨") label, denoting that the set of its children is a *disjunctive set*.

A user operates interactively and iteratively on a faceted interface while building a faceted query. The user can refine the query by operating on this interface and by browsing and exploring the application domain modeled by the underlying fac-ontology. The operations on a faceted interface are listed in the context menu shown in Fig. 4.

Using the context menu, a user can set or change labels assigned to nodes, as well as to enlarge or reduce the interface. The set of labels defines the *state* of a node. A state is a quintuple determined by functions defined in Definition 8, i.e., $Sel()$, $AndOr()$, $PosNeg()$, $NumRestr()$, and $Self()$.

In addition, the following operations are used to modify the content of a faceted interface:

1. $AttributeValues(v)$—is applicable to data property nodes to upload (from a database), insert, edit or remove values of this data property.
2. $Clone(v)$ *(duplicate)*—allows to clone the subtree rooted in the indicated node. A new subtree is created and inserted as a child of the indicated node, and the inserted subtree is isomorphic, up to node identifiers, with the cloned subtree. Applicable to class- and property nodes except of the *root* node.
3. $Explore(v)$—allows to display invisible parts of the fac-ontology and include them into the faceted interface.

4. *RemoveALLUnchecked*(v)—removes from the faceted interface all unselected (unchecked) nodes.

A graphical form of a state of a sample faceted interface is shown in Fig. 2. It arises from Fig. 3 by: adding one subtree as a result of using the *Explore(Person)* operation, adding a number restriction to $authorOf$ node, negating *affiliation* node, and labeling by *Self* the *authorConfPCMember* node.

Proposition 1 *A faceted interface FI determines an expression F with the syntax defined by the grammar:*

$$\begin{aligned}
F &::= \Diamond root\{U_1, \ldots, U_n\} \mid \neg F,\\
U &::= A \mid \Diamond A(\{T_1, \ldots, T_n\}) \mid \neg U,\\
T &::= P \mid Self\,P \mid \Diamond P\{U_1, \ldots, U_n\} \mid P\{a_1, \ldots, a_n\}\\
&\quad \mid (\geq k)\Diamond P\{U_1, \ldots, U_n\} \mid (\geq k)P\{a_1, \ldots, a_n\} \mid \neg T.
\end{aligned}$$

Proof According to Definition 8 p. (2), the DFS serialization of FI converts it to a faceted view, i.e., a nested facet conforming to the syntax given in Definition 6. Additionally, the subexpressions of this nested facet can be labeled by the negation symbol ($\neg$), number restriction ($\geq k$) or by the local reflexivity symbol (*Self*). □

5 Faceted queries over ontological databases

5.1 Syntax and semantics of faceted queries—faceted normal form (FacNF) of queries

Any faceted interface in which all nodes are selected (i.e., all unselected are removed) defines a *faceted query*. A faceted interface imposes a tree-shaped structure on a faceted query and: (a) the negation symbol precedes a single node (not a set of nodes), (b) the set of all children of a node is: (i) either conjunctive or disjunctive and (ii) all children are of the same category, i.e., all are class nodes, property nodes or constant nodes.

Definition 9 A *faceted query* is a faceted interface in which every node is selected.

The semantics of a faceted query F is defined by translating it into a description logic expression. We assume that $root$ is the *universal reflexive property* with the semantics: $root^{\mathcal{I}} = \{(x, x) \mid x \in \Delta^{\mathcal{I}}\}$.

Definition 10 The semantics of a faceted query F is a query $q = \tau(F)$, where the translation function $\tau()$ is defined as:

$$\begin{aligned}
\tau(\Diamond root\{U_1, \ldots, U_n\}) &= \exists root.\Diamond\{U_1, \ldots, U_n\} =\\
&\quad \Diamond\{U_1, \ldots, U_n\},\\
\tau(A) &= A,\\
\tau(\Diamond A\{T_1, \ldots, T_n\}) &= A \sqcap (\Diamond\{T_1, \ldots, T_n\}),\\
\tau(P) &= \exists P,\\
\tau(P\{a_1, \ldots, a_n\}) &= \exists P.\{a_1, \ldots, a_n\},\\
\tau(\Diamond P\{U_1, \ldots, U_n\}) &= \exists P.\Diamond\{U_1, \ldots, U_n\}.\\
\tau(Self\,P) &= \exists P.Self,\\
\tau((\geq k)\Diamond P\{U_1, \ldots, U_n\}) &= (\geq k)P.\Diamond\{\tau(U_1), \ldots, \tau(U_n)\},\\
\tau((\geq k)P\{a_1, \ldots, a_n\}) &= (\geq k)P.\{a_1, \ldots, a_n\},\\
\tau(\neg X) &= \neg\tau(X),
\end{aligned}$$

where X denotes any (not negated) expression of categories F, U, T (see Proposition 1).

A $\mathcal{SROIQ}^{Fac}$ query with the syntax conforming to the syntax of faceted queries will be called a query in a *faceted normal form* (FacNF). The following proposition follows from Definition 10.

Proposition 2 *A faceted query conforms to the syntax specified by the following grammar called a faceted normal form (FacNF):*

$$\begin{aligned}
q &::= G \mid \neg G\\
G &::= E_1 \sqcup \cdots \sqcup E_n \mid E_1 \sqcap \cdots \sqcap E_n\\
E &::= A \mid A \sqcap (D_1 \sqcup \cdots \sqcup D_n) \mid\\
&\quad A \sqcap (D_1 \sqcap \cdots \sqcap D_n) \mid \neg E\\
D &::= \exists P \mid \exists P.Self \mid \exists P.G \mid (\geq k)P.G \mid\\
&\quad \exists P.\{a_1, \ldots, a_n\} \mid (\geq k)P.\{a_1, \ldots, a_n\} \mid \neg D
\end{aligned}$$

Example 3 The following $\mathcal{SROIQ}^{Fac}$ queries are not in FacNF: $q_1 = A_1 \sqcap (A_2 \sqcup A_3)$, $q_2 = \exists P_1.\neg(A_1 \sqcap A_2)$, $q_3 = \exists P_2.\{a\} \sqcap \neg A_1$. In Example 4, they will be transformed into FacNFs.

5.2 Transformation of $\mathcal{SROIQ}^{Fac}$ into FacNF

We will show that the expressive power of the faceted query system is equal to that of $\mathcal{SROIQ}^{Fac}$. To this end, we will show that every $\mathcal{SROIQ}^{Fac}$ query can be transformed into a query in FacNF. This means that for any query in $\mathcal{SROIQ}^{Fac}$ there is a semantically equivalent query that can be formulated using the faceted interface.

Theorem 1 *Every query in $\mathcal{SROIQ}^{Fac}$ over a fac-ontology $\mathcal{O} = (\mathcal{V}, \mathcal{C}, \mathcal{A})$ can be transformed into an equivalent query in FacNF over $\mathcal{O}$.*

Proof Let q be a query in $\mathcal{SROIQ}^{Fac}$ over $\mathcal{O}$. We transform q into FacNF as follows:

1. Every class A occurring in q is replaced by the left-hand side of its specialization (unfolding the specialization). If $\mathcal{O} \models \top^A \sqcap C \equiv A$, then

   $$q_{spec} = q.replace(A, \top^A \sqcap C),$$

   and q_{spec} arises from q by replacing A with $\top^A \sqcap C$, and syntax of C is given in Definition 1.
2. q_{spec} is converted into disjunctive normal form (DNF):

   $$q_{dnf} = dnf(q_{spec}) = q_1 \sqcup \cdots \sqcup q_m,$$

   where q_i, $1 \le i \le m$ is a conjunction of queries in $\mathcal{SROIQ}^{Fac}$.
3. Every disjunct q_i, $1 \le i \le m$, containing negation of a top class is removed from q_{dnf}, since it evaluates to the empty set. The reduced disjunction is:

   $$q_{red} = reduce(q_{dnf}) = q_1 \sqcup \cdots \sqcup q_n.$$

4. Every disjunct q_i, $1 \le i \le n$, in q_{red} can be rewritten into the form:

   $$q_i^{FacNF} = type(q_{red}) \sqcap (D_1^i \sqcap \cdots \sqcap D_{k_i}^i),$$

   where D_j^i, $1 \le i \le k_i$ conforms to the syntax of D in Proposition 2. Then, q_i^{FacNF} is in FacNF, and $q_{red} \equiv q_{red}^{FacNF}$, where q_{red}^{FacNF} is in FacNF, and

   $$q_{red}^{FacNF} = q_1^{FacNF} \sqcup \cdots \sqcup q_n^{FacNF}.$$

5. The procedure is applied recursively to every subquery q appearing in formulas of the form $\exists P.q$.

□

Example 4 Let q_1, q_2 and q_3 be queries specified in Example 3. Let classes A_1, A_2, and A_3 be defined by the following specializations:

$A_0 \sqcap \exists P_1'.A_1' \equiv A_1$, $A_0 \sqcap \exists P_2'.A_2' \equiv A_2$, $A_0 \sqcap \exists P_3'.A_3' \equiv A_3$, where

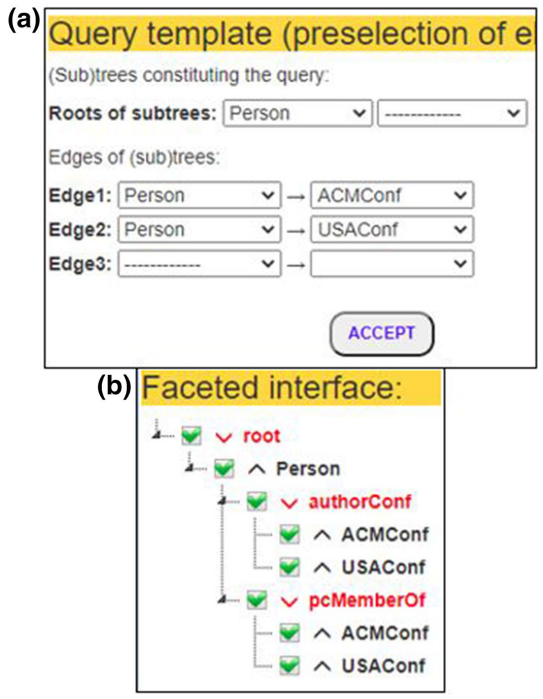
$A_0 = type(A_1) = type(A_2) = type(A_3) = type(\exists P_2)$, $A_* = type(\exists P_1)$. Then:

1. Transforming $q_1 = A_1 \sqcap (A_2 \sqcup A_3)$ into FacNF:

   $$FacNF(q_1) = A_0 \sqcap (\exists P_1'.A_1' \sqcap \exists P_2'.A_2') \\ \sqcup A_0 \sqcap (\exists P_1'.A_1' \sqcap \exists P_3'.A_3').$$

2. Transforming $q_2 = \exists P_1.\neg(A_1 \sqcap A_2)$ into FacNF:

   $$FacNF(q_2) = A_* \sqcap \exists P_1.(\neg A_1 \sqcup \neg A_2).$$


(a) Query template (preselection of e

(Sub)trees constituting the query:

Roots of subtrees: Person ▾ | ------------ ▾

Edges of (sub)trees:

Edge1: Person ▾ → ACMConf ▾

Edge2: Person ▾ → USAConf ▾

Edge3: ------------ ▾ → ▾

ACCEPT

(b) Faceted interface:

- ✓ root
  - ✓ Person
    - ✓ authorConf
      - ✓ ACMConf
      - ✓ USAConf
    - ✓ pcMemberOf
      - ✓ ACMConf
      - ✓ USAConf

Fig. 5 A sample of: **a** a query template and **b** a faceted interface generated for it

3. Transforming $q_3 = \exists P_2.\{a\} \sqcap \neg A_1$ into FacNF:

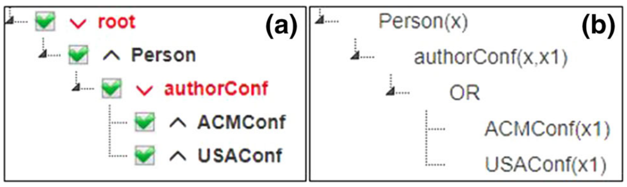
$$q_{spec}^3 = \exists P_2.\{a\} \sqcap \neg(A_0 \sqcap \exists P_1'.A_1'),$$
$$q_{dnf}^3 = \exists P_2.\{a\} \sqcap \neg A_0 \sqcup \exists P_2.\{a\} \sqcap \neg\exists P_1'.A_1',$$
$$q_{red}^3 = \exists P_2.\{a\} \sqcap \neg\exists P_1'.A_1',$$
$$FacNF(q_3) = A_0 \sqcap (\exists P_2.\{a\} \sqcap \neg\exists P_1'.A_1').$$

6 Faceted query formulation in DAFO

6.1 Examples of query formulation

Now, we show how some representative queries can be formulated in the DAFO system [24].

Below, we consider queries, some of which concern the involvement of people in conferences—ACM conferences and/or conferences in the USA. Thus, a user can start with indicating the relevant classes to the intended query (as a query template, Fig. 5a): *Person* as the type of the expected answers, as well as *ACMConf* and *USAConf* related somehow with the *Person*. In response, a faceted interface is created, (Fig. 5b). Note that a person can be connected to conferences as a participant and/or as a PC member. Formulation of a query over a faceted interface requires a sequence of operations on this interface. Each state of a faceted interface represents a faceted query, so operations over the interface are transformations in a space of faceted queries.



```
(a)
root
  Person
    authorConf
      ACMConf
      USAConf
```

```
(b)
Person(x)
  authorConf(x,x1)
    OR
      ACMConf(x1)
      USAConf(x1)
```

Fig. 6 Query q_1 (**a**) and its first-order syntax tree **b**

We will consider the following kinds of (faceted) queries:

1. Positive existential queries:
   - with default conjunctive and disjunctive facets—query q_1,
   - a disjunctive set is switched to a conjunctive one—query q_2,
   - a subtree is cloned (duplicated) and values of some data properties are added—query q_3.
2. Queries with negation:
   - query with one negation (exclusion)—query q_4,
   - query with double negation (equivalent to a universal quantification)—query q_5.
3. Query with a number restriction—query q_6,
4. Query with a local reflexivity (a cycle)—query q_7.

For every query we provide:

- a natural language version,
- a DL version in $\mathcal{SROIQ}^{Fac}$,
- graphical forms of faceted queries (q_1)–(q_7), their first-order forms, for (q_1)–(q_5), and a notation involving variables for (q_6) and (q_7)—all as screenshots on DAFO,
- operations on the faceted interface used to create the final faceted query.

1. Positive existential queries

q_1: *"Authors of papers presented at an ACM conference or at a conference in the USA"*

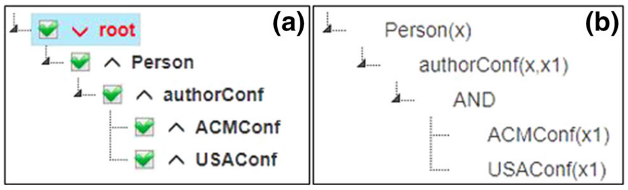
$$q_1 = Person \sqcap \exists authorConf.(ACMConf \sqcup USAConf).$$

In Fig. 6a, the query is expressed as a faceted query, and in Fig. 6b there is a first-order form of q_1. The query is created by performing the following sequence of operations on the faceted interface in Fig. 5b:

```
pcMemberOf.Uncheck(); RemoveAllUnchecked().
```

q_2: *"Authors of papers presented at an ACM conference in the USA"*

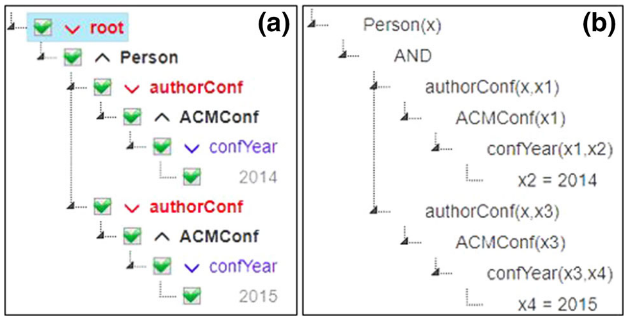
$$q_2 = Person \sqcap \exists authorConf.(ACMConf \sqcap USAConf).$$



```
(a)
root
  Person
    authorConf
      ACMConf
      USAConf
```

```
(b)
Person(x)
  authorConf(x,x1)
    AND
      ACMConf(x1)
      USAConf(x1)
```

Fig. 7 Query q_2 (**a**) and its first-order syntax tree (**b**)



```
(a)
root
  Person
    authorConf
      ACMConf
        confYear
          2014
    authorConf
      ACMConf
        confYear
          2015
```

```
(b)
Person(x)
  AND
    authorConf(x,x1)
      ACMConf(x1)
        confYear(x1,x2)
          x2 = 2014
    authorConf(x,x3)
      ACMConf(x3)
        confYear(x3,x4)
          x4 = 2015
```

Fig. 8 Query q_3 (**a**) and its first-order syntax tree (**b**)

In Fig. 7, the query is depicted as a faceted query and its first-order form as a result of operating on the query/faceted interface in Fig. 6a:

```
authorConf.SetToAND().
```

q_3: *"Authors who presented her/his papers at ACM conferences in years 2014 and 2015"*

$$\begin{aligned} q_3 = {} & Author \sqcap \\ & \exists authorConf.(ACMConf \sqcap \exists confYear.\{'2014'\}) \sqcap \\ & \exists authorConf.(ACMConf \sqcap \exists confYear.\{'2015'\}) \end{aligned}$$

Query q_3 is formulated taking Fig. 6a as a current form of a faceted interface. The following operations are performed on it:

```
USAConf.Uncheck(); ACMConf.Expand();
confYear.Check();
confYear.AttributeValue.AddValue('2014');
RemoveALLUnchecked();
AuthorConf.ClonSubtree();
confYear[2].AttributeValue.EditValue
('2014'/'2015');
```

A result of these operations is shown in Fig. 8.

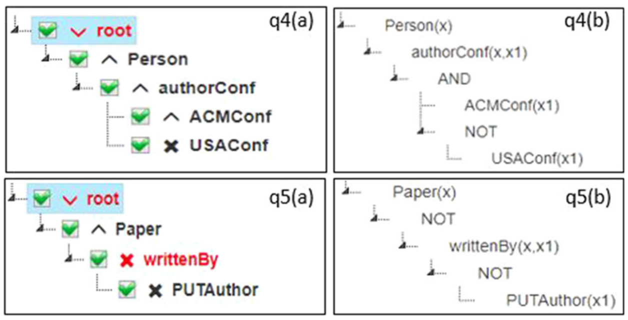


Fig. 9 Queries q_4 and q_5 and their first-order syntax trees

2. Queries with negations

q_4: *"Authors of papers presented at ACM conferences but NOT in the USA"*

$$q_4 = Person \sqcap \exists authorConf.(ACMConf \sqcap \neg USAConf).$$

q_4 arises from q_2 by excluding the *USAConf*:

```
USAConf.Exclude();
```

The result is in Fig. 9–q4(a) and q4(b). In q_5, negation is used to express a universal quantification.
q_5: *"Papers written only by PUTAuthors"*

$$q_5 = Paper \sqcap \neg\exists writtenBy.\neg PUTAuthor) = Paper \sqcap \forall writtenBy.PUTAuthor.$$

Formulation of q_5 requires double exclusion (double negation) (Fig. 9)—q5(a) and q5(b).

3. Queries with number restrictions

q_6: *"Authors participating in over ten ACM conferences in the USA"*

$$q_6 = Person \sqcap (> 10) authorConf.(ACMConf \sqcap USAConf).$$

The query is formulated in DAFO as shown in Fig. 10. The argument of $count(x_1)$ indicates that x_1 is tested for the required number of distinct values. If x is connected to more than 10 different valuations of x_1, then this valuation of x is returned as an answer. To formulate the query, we use the query in Fig. 7a as a faceted interface and perform the following operation on it:

```
authorConf.SetNumRestr.count()>10;
```

4. Queries with local reflexivity (looking for cycles).

q_7: *"Authors of papers presented at conferences where the author was a PC member"*

$$q_7 = Person \sqcap \exists authConfPCMember.Self,$$

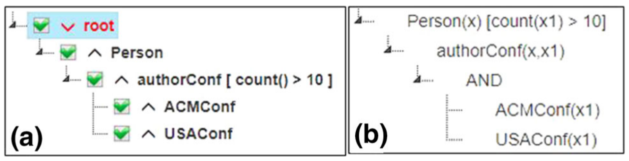


Fig. 10 Query q_6 (**a**), and its syntax tree (**b**)

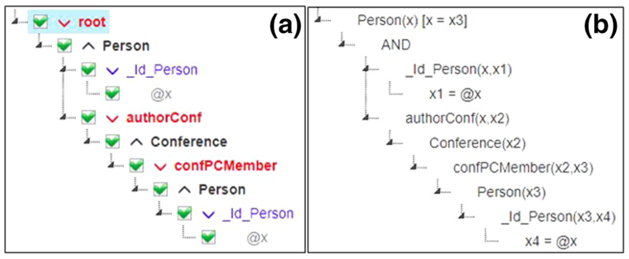


Fig. 11 Query q_7 (**a**) and its syntax tree (**b**)

Table 2 Expressiveness of faceted interfaces in four systems

Query	DAFO	BrowseRDF	Sewelis	SemFacet
A	+	+	+	+
$q_1 \sqcup q_2$	+	−	+	+
$q_1 \sqcap q_2$	+	+	+	+
$q_1 \sqcap \neg q_2$	+	+	+	−
$\neg q$	+	+	+	−
$\exists R.q,$	+	+	+	+
$\exists R$	+	+	+	+
$\exists R.\{a\}$	+	+	+	+
$(\geq k)\exists R.q$	+	−	−	+
$(\geq k)\exists R$	+	−	−	+
$\exists R.Self$	+	+	+	−

where:
$authorConf \circ confPCMember \equiv authConfPCMember$.

The query is formulated in DAFO as shown in Fig. 11a. A syntax tree in Fig. 11b is an intermediate form with variables and a global variable $@x$ that indicates objects which are connected with themselves by the property $authorConf \circ confPCMember$. The occurrences of $@x$ determine the equality $[x = x_3]$ saying that we are looking for objects assigned to x and x_3 that are equal.

6.2 Expressiveness of DAFO compared to other systems

In Table 2, we compare the expressive power of four faceted query systems: DAFO, BrowseRDF [53], Sewelis [34] and SemFacet [7,64].

The first column in the table contains queries in $\mathcal{SROIQ}^{Fac}$, which are used as reference points to interpret the semantics of operations in the compared systems.

Since there are differences in the interpretations of some concepts and notions (especially, negation, aggregation, and recursion), we describe these differences in comments.

1. *Disjunction and conjunction.* In DAFO, each disjunction $q_1 \sqcup q_2$, and conjunction $q_1 \sqcap q_2$, requires that q_1 and q_2 are of the same type.
2. *Negation* A negation is computed as the complement with respect to a *guard.*
   - In a negation $q_1 \sqcap \neg q_2$, the guard is q_1, and q_1 and q_2 are of the same type.
   - In DAFO, a negation $\neg q$ is *guarded* by the $type(q)$.
   - In BrowseRDF, the negation can only concern the existence of an property. Negation is true for objects that do not have the negated property.
   - In Sewelis, the complement is computed with respect to the set of all objects (the universal class).
3. *Navigation, recursion, reachability*. In all analyzed systems, existential restrictions: $\exists R.q$, $\exists R$, $\exists R.\{a\}$, are fundamental for navigation through the underlying ontology, where: (a) $\exists R.q$ denotes a set of objects connected via an object property R with objects in q, (b) $\exists R$ denotes a set of objects having a property R (irrespective of its value), $\exists R.\{a\}$ is a set of objects connected via a data property R with a constant a. R can denote a property P or its inversion P^- (only for object properties). Properties can be recursively composed, expressing in this way a (restricted) form of a recursion. In DAFO, composed properties can be defined as *chains* of other properties. In SemFacet, so called *reachability atoms*, $\mathsf{Next}(x, y)$ and $\mathsf{Next}^+(x, y)$ are introduced. They denote (dynamically) a property, or a sequence of properties, leading from x to y.
4. *Aggregation.* In $\mathcal{SROIQ}$, an aggregation is limited to a *number restriction*. We also do this in DAFO restricting ourselves to the *count*() function.
5. *Cycles.* A reflexivity restriction $\exists R.Self$, denotes objects which are connected via R with themselves. In this way, cycles can be found. In DAFO and in Sewelis, this is achieved by means of special variables (in DAFO prefixed by @). Two different occurrences in a query of the same variable $@x$ indicate that a value of $@x$ is connected with itself by a property (possibly composed) specified in the query.

7 Materialization of constraint rules

Given a fac-ontology $\mathcal{O} = (\mathcal{V}, \mathcal{C}, \mathcal{A})$, our goal is to convert $\mathcal{O}$ into such an ontological database $\mathcal{O}_{db} = (\mathcal{V}, \mathcal{C}, \mathcal{A}_{db})$ that $\mathcal{A}_{db}$ is a minimal model of $\mathcal{A} \cup \mathcal{C}$, i.e., $\mathcal{A}_{db} \models \mathcal{A} \cup \mathcal{C}$. Such the $\mathcal{A}_{db}$ can be obtained as the *chase* of $\mathcal{A}$ with respect to $\mathcal{C}$, $\mathcal{A}_{db} = chase_{\mathcal{C}}(\mathcal{A})$. In other words, $\mathcal{A}$ is included in $\mathcal{A}_{db}$ and $\mathcal{C}$ is *materialized* in $\mathcal{A}_{db}$.

In general, the chase procedure can: (a) be infinite, (b) terminate with the fail, (c) produce a finite set of facts containing $\mathcal{A}$ and all consequences of $\mathcal{C}$. Some rules in $\mathcal{C}$, namely disjointness and functionality, are used to verify consistency of the ontological database. The functionality rules can also be used to discover some missing values, represented by labeled nulls.

We divide $\mathcal{C}$ into two subsets: $\mathcal{C}_1$ and $\mathcal{C}_2$:

1. $\mathcal{C}_1$ contains rules of the form: $C_1 \sqsubseteq C_2$ and $R_1 \sqsubseteq R_2$. Their first-order forms are *tuple-generating dependencies*:
   - $\forall \mathbf{x}, \mathbf{y}(\varphi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z} \psi(\mathbf{x}, \mathbf{z}))$,

   where $\mathbf{x}, \mathbf{y}, \mathbf{z}$ are tuples of variables and $\varphi(\mathbf{x}, \mathbf{y})$, $\psi(\mathbf{x}, \mathbf{z})$ are conjunctions of atoms over all the given variables and constants. These rules are used to chase new facts.
2. $\mathcal{C}_2$ contains disjointness rules, $A_1 \sqsubseteq \neg A_2$, $R_1 \sqsubseteq \neg R_2$, and functionality rules ($\mathsf{funct}\ S$), with first-order forms, respectively:
   - $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \neg \psi(\mathbf{x}))$,
   - $\forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow x_1 = x_2)$, where $x_1, x_2 \in \mathbf{x}$.

   These rules are mainly used to check if the chased set of facts is consistent.

Chasing with respect to: $\sigma = \forall \mathbf{x}, \mathbf{y}(\varphi(\mathbf{x}, \mathbf{y}) \rightarrow \exists \mathbf{z} \psi(\mathbf{x}, \mathbf{z}))$

Let ω be a valuation, $\omega : \mathbf{x} \cup \mathbf{y} \mapsto \mathsf{Const}$, such that $\mathcal{A} \models \varphi(\omega(\mathbf{x}), \omega(\mathbf{y}))$. Then:

1. If $\mathcal{A} \not\models \exists \mathbf{z} \psi(\omega(\mathbf{x}), \mathbf{z})$, then as a result of *applying σ to $\mathcal{A}$ with valuation ω* we obtain the set $\mathcal{A}'$ that extends $\mathcal{A}$ with facts $A(\omega'(v_1))$ and $R(\omega'(v_2), \omega'(v_3))$, where $A(v_1)$ and $R(v_2, v_3)$ occur in $\psi(\mathbf{x}, \mathbf{z})$, $v_i \in \mathbf{x} \cup \mathbf{z}$, for $i = 1, 2, 3$, and: (a) if $v_i \in \mathbf{x}$, then $\omega'(v_i) = \omega(v_i)$; (b) if $v_i \in \mathbf{z}$, then $\omega'(v_i)$ is a fresh labeled null N. We denote this as $\mathcal{A} \xrightarrow{\sigma,\omega} \mathcal{A}'$.
2. If $\mathcal{A} \models \exists \mathbf{z} \psi(\omega(\mathbf{x}), \mathbf{z})$, then σ is not applicable to $\mathcal{A}$ with the valuation ω.

Chasing with respect to: $\sigma = \forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow \neg \psi(\mathbf{x}))$.

Let $\omega : \mathbf{x} \mapsto \mathsf{Const}$ be a valuation such that $\mathcal{A} \models \varphi(\omega(\mathbf{x}))$. Then:

1. If $\mathcal{A} \models \psi(\omega(\mathbf{x}))$, then the *result of applying σ to $\mathcal{A}$ with* ω is "failure," which is denoted by $\mathcal{A} \xrightarrow{\sigma,\omega} \mathsf{FAIL}$.
2. If $\mathcal{A} \not\models \psi(\omega(\mathbf{x}))$, then σ is not applicable to $\mathcal{A}$ with the valuation ω.

Chasing with respect to: $\sigma = \forall \mathbf{x}(\varphi(\mathbf{x}) \rightarrow x_1 = x_2)$.

Let $\omega : \mathbf{x} \mapsto \mathsf{Const}$ be a valuation such that $\mathcal{A} \models \varphi(\omega(\mathbf{x}))$. Then,

1. if $\omega(x_1) \neq \omega(x_2)$ and neither $\omega(x_1)$ nor $\omega(x_2)$ is a labeled null, then $\mathcal{A} \xrightarrow{\sigma,\omega} \mathsf{FAIL}$.
2. if $\omega(x_1)$ is a labeled null N, then $\mathcal{A}'$ is obtained from $\mathcal{A}$ by replacing every occurrence of N in $\mathcal{A}$ by $\omega(x_2)$, denoted by $\mathcal{A} \xrightarrow{\sigma,\omega} \mathcal{A}'$.

7.1 Termination of chase

The chase can be defined as the *data exchange* problem [21, 33], and can be understood as a pair $(\mathbf{T}, \mathcal{C})$, where $\mathbf{T} = Sig(\mathcal{C}) \cup \mathsf{Const}$ is a target schema, and $\mathcal{C}$ is a set of target-to-target dependencies. A *solution* of an instance $\mathcal{A}$ of $\mathbf{T}$ with respect to $\mathcal{C}$ is an instance $\mathcal{A}'$ of $\mathbf{T}$ such that $\mathcal{A}' \models \mathcal{A} \cup \mathcal{C}$. In general, an instance $\mathcal{A}$ can have infinitely many solutions. In particular, if $\mathcal{A}'$ is a solution for $\mathcal{A}$ with respect to $\mathcal{C}$, then every $\mathcal{A}''$ containing $\mathcal{A}'$ is also a solution for $\mathcal{A}$. A set $\mathcal{A}'$ of facts is the *universal solution* of $\mathcal{A}$ with respect to $\mathcal{C}$ if (a) $\mathcal{A}'$ is a solution of $\mathcal{A}$, and (b) for each solution $\mathcal{A}''$ of $\mathcal{A}$ there is a homomorphism h from $\mathcal{A}'$ to $\mathcal{A}''$ (h is the identity on constants). Intuitively, a universal solution contains no more and no less information than that specified by the given data exchange problem.

The chase procedure $chase_{\mathcal{C}}(\mathcal{A})$ is guaranteed to terminate in polynomial time producing the universal solution for $\mathcal{A}$ with respect to $\mathcal{C}$ if the *dependency graph* of $\mathcal{C}$ is *weakly acyclic* [5,33].

Definition 11 The *dependency graph*, $DG(\mathcal{C})$, over a set $\mathcal{C}$ of constraint rules is constructed as follows: (a) for every class A occurring in $\mathcal{C}$ there is a node $(A, 1)$ in $DG(\mathcal{C})$; (b) for every property R occurring in $\mathcal{C}$ there are two node $(R, 1)$ and $(R, 2)$ in $DG(\mathcal{C})$; (c) for every rule of the form $A_1 \sqsubseteq \exists R.A_2$ in $\mathcal{C}$ there are three edges in $DG(\mathcal{C})$:

– $(A_1, 1) \rightarrow (R, 1)$, $(R, 2) \rightarrow (A_2, 1)$ – *regular* edges,
– $(A_1, 1) \xrightarrow{*} (R, 2)$ – a *special* edge (labeled by $*$).

$\mathcal{C}$ is *weakly acyclic* if its dependency graph $DG(\mathcal{C})$ does not have any *weak cycle*, i.e., a cycle going through a special edge.

Sets of constraint rules in fac-ontologies usually have a lot of weak cycles because properties and their inverses are usually taken into account.

Example 5 Let $\mathcal{C}_a = \{\sigma_1, \sigma_2\}$, where

$$\sigma_1 = Author \sqsubseteq \exists authorOf.Paper,$$
$$\sigma_2 = Paper \sqsubseteq \exists writtenBy.Author.$$

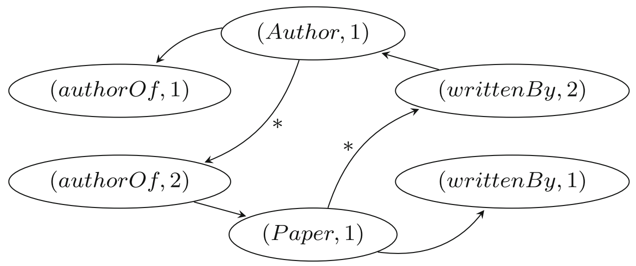


Fig. 12 The dependency graphs for $\mathcal{C}_a$ in Example 5 has a weak cycle: $(Author, 1) \xrightarrow{*} (authorOf, 2) \rightarrow (Paper, 1) \xrightarrow{*} (writtenBy, 2) \rightarrow (Author, 1)$

The dependency graph of $\mathcal{C}_a$ is depicted in Fig. 12. The graph has a weak cycle, and this leads to an infinite chase. On the other hand, a finite solution exists, but this solution is not the universal solution. We see that

$$\mathcal{A}' = \{Author(a), authorOf(a, \mathsf{N}_1), Paper(\mathsf{N}_1), writtenBy(\mathsf{N}_1, a)\},$$

is a solution for $\mathcal{A}$. However, a solution is also

$$\mathcal{A}'' = \{Author(a), authorOf(a, \mathsf{N}_1), Paper(\mathsf{N}_1), writtenBy(\mathsf{N}_1, \mathsf{N}_2), Author(\mathsf{N}_2), authorOf(\mathsf{N}_2, \mathsf{N}_3), Paper(\mathsf{N}_3), writtenBy(\mathsf{N}_3, a)\}.$$

However, there is no any homomorphism $h : \mathcal{A}' \rightarrow \mathcal{A}''$ since h is not identity on a, because $h(a) = a$ and $h(a) = \mathsf{N}_2$. In this case, the universal solution for $\mathcal{A}$ with respect to $\mathcal{C}_1$ does not exist.

7.2 Reflexive weak cycles

Now, we identify a class of weak cycles in dependency graphs and call them *reflexive weak cycles* (RWC). For RWC, we will modify the chase problem. The modified chase will terminate with a universal solution. This universal solution is also a solution (but not a universal solution) for the chase before the modification.

Definition 12 Let $\mathcal{C}$ be a set of constraint rules and $DG(\mathcal{C})$ a dependency graph over $\mathcal{C}$. A *reflexive weak cycle* (RWC) in $DG(\mathcal{C})$ is a cycle of the form

$$(A_1, 1) \xrightarrow{*} (R_1, 2) \rightarrow (A_2, 1) \xrightarrow{*} \cdots \xrightarrow{*} (R_n, 2) \rightarrow (A_1, 1),$$

such that:

1. Every subpath $(A_i, 1) \xrightarrow{*} (R_i, 2) \rightarrow (A_{i+1}, 1)$ is implied by the rule $A_i \sqsubseteq \exists R_i.A_{i+1} \in \mathcal{C}$, $1 \leq i \leq n$, $n \geq 2$, $A_{n+1} = A_1$,
2. $\mathcal{C} \models R_1 \circ \cdots \circ R_{n-1} \sqsubseteq R_n^-$.

A RWC is abbreviated as $(A_1, R_1, A_2, \ldots, A_n, R_n, A_1)$.

Proposition 3 *Let $dom(R_1) = A$ and $R_1 \circ \cdots \circ R_{n-1} \sqsubseteq R_n^-$. Then, $A \equiv \exists(R_1 \circ \cdots \circ R_n).Self$.*

Proof We prove the proposition for $n = 2$. Let us assume that $R_1 \sqsubseteq R_2^-$ and $dom(R_1) = A$.
$\Rightarrow$ If $A(a)$ then for some b, $R_1(a, b)$ and $R_2(b, a)$. Therefore, $(\exists(R_1 \circ R_2).Self)(a)$.
$\Leftarrow$ Now, let $(\exists(R_1 \circ R_2)Self)(c)$. Then, for some d, $R_1(c, d)$ and $R_2(d, c)$. Thus, $A(c)$.
This reasoning can easily be extended to any $n > 2$. □

Proposition 3 shows that the chain $R_1 \circ \cdots \circ R_n$ of properties belonging to RWC is "local reflexive," i.e., any object in the domain of R_1 is connected with itself via this chain. This property justifies the term *"reflexive"* for the class of weak cycles under consideration.

Example 6 Let σ_1 and σ_2 be defined in Example 5, and $\sigma_2' = Paper \sqsubseteq \exists reviewedBy.Author$. Let $\mathcal{C}' = \{\sigma_1, \sigma_2\}$, and $\mathcal{C}'' = \{\sigma_1, \sigma_2'\}$. Then, both:

$p_1 = (Author, 1) \overset{*}{\rightarrow} (authorOf, 2) \rightarrow (Paper, 1) \overset{*}{\rightarrow} (writtenBy, 2) \rightarrow (Author, 1)$, and

$p_2 = (Author, 1) \overset{*}{\rightarrow} (authorOf, 2) \rightarrow (Paper, 1) \overset{*}{\rightarrow} (reviewedBy, 2) \rightarrow (Author, 1)$ are weak cycles in, respectively, $DG(\mathcal{C}')$ and $DG(\mathcal{C}'')$. But only p_1 is RWC since $authorOf \sqsubseteq writtenBy^-$.

We define a *p-aware chase* as a chase in the presence of a RWC p. The aim is that the *p-aware* chase terminates with a solution, although not a universal solution.

Definition 13 Let $\mathcal{C}$ be a set of constraint rules, $\mathcal{A}$ a set of facts over $Sig(\mathcal{C}) \cup \mathsf{Const}$, and $p = (A_1, R_1, A_2, \ldots, A_n, R_n, A_1)$ a RWC over $\mathcal{C}$. A *p-aware chase* of $\mathcal{A}$ with respect to $\mathcal{C}$, $chase_{\mathcal{C}}^{\{p\}}(\mathcal{A})$, is a chase $chase_{\mathcal{C}'}(\mathcal{A})$, such that $\mathcal{C}' = \mathcal{C} \setminus \{A_n \sqsubseteq \exists R_n.A_1\}$.

Intuitively, our goal is to prevent firing of the rule $A_n \sqsubseteq \exists R_n.A_1$. Instead, the rule $R_1 \circ \cdots \circ R_{n-1} \sqsubseteq R_n^-$ is applied.

Example 7 Let $\mathcal{C}_b = \{\sigma_1, \sigma_2, \sigma_3\}$, σ_1, σ_2 be defined in Example 5, and $\sigma_3 = authorOf \sqsubseteq writtenBy^-$. Then,

$$p = (Author, authorOf, Paper, writtenBy, Author)$$

is the weak cycle presented in Fig. 12. Because of σ_3, p is also a RWC. Let

$$\mathcal{A} = \{Author(a)\}.$$

Then a *p-aware* chase, $chase_{\{\sigma_1,\sigma_2,\sigma_3\}}^{\{p\}}(\mathcal{A})$, is the chase

$$chase_{\{\sigma_1,\sigma_3\}}(\mathcal{A}).$$

First-ordered forms of the rules involved in the chase are:

$$\sigma_1 = \forall x(Author(x) \rightarrow \exists y\, authorOf(x, y) \wedge Paper(y)),$$
$$\sigma_3 = \forall x, y(authorOf(x, y) \rightarrow writtenBy(y, x)).$$

The *p*-aware chase consists of the following steps:

$$\{Author(a)\} \xrightarrow{\sigma_1, [x \mapsto a]} \{Author(a), authorOf(a, \mathsf{N}_1), Paper(\mathsf{N}_1)\} \xrightarrow{\sigma_3, [x \mapsto a, y \mapsto \mathsf{N}_1]} \{Author(a), authorOf(a, \mathsf{N}_1), Paper(\mathsf{N}_1), writtenBy(\mathsf{N}_1, a)\} = \mathcal{A}'.$$

$\mathcal{A}'$ is a *universal solution* for $\mathcal{A}$ with respect to $\{\sigma_1, \sigma_3\}$, and $\mathcal{A}'$ is also a *solution* for $\mathcal{A}$ with respect to $\{\sigma_1, \sigma_2, \sigma_3\}$, but not a universal solution. To show that $\mathcal{A}'$ is not a universal solution for $\mathcal{A}$ with respect to $\{\sigma_1, \sigma_2, \sigma_3\}$, let us note that, for example,

$$\mathcal{A}''' = \{Author(a), authorOf(a, \mathsf{N}_1), Paper(\mathsf{N}_1), writtenBy(\mathsf{N}_1, \mathsf{N}_2), Author(\mathsf{N}_2), authorOf(\mathsf{N}_2, \mathsf{N}_3), Paper(\mathsf{N}_3), writtenBy(\mathsf{N}_3, \mathsf{N}_2), writtenBy(\mathsf{N}_1, a)\}.$$

is also a solution for $\mathcal{A}$ with respect to $\{\sigma_1, \sigma_2, \sigma_3\}$, but there is not a homomorphism $h : \mathcal{A}' \rightarrow \mathcal{A}'''$ preserving constants (we have $h(a) = a$ and $h(a) = \mathsf{N}_2$).

Theorem 2 *Let $p = (A_1, R_1, A_2, \ldots, A_n, R_n, A_1)$ be a RWC over a set $\mathcal{C}$ of constraint rules. Let $chase_{\mathcal{C}}^{\{p\}}(\mathcal{A}) = chase_{\mathcal{C}'}(\mathcal{A}) = \mathcal{A}'$, $\mathcal{C}' = \mathcal{C} \setminus \{A_n \sqsubseteq \exists R_n.A_1\}$, be a p-aware chase. Then $\mathcal{A}'$ is a universal solution for $\mathcal{A}$ with respect to $\mathcal{C}'$ and a solution for $\mathcal{A}$ with respect to $\mathcal{C}$.*

Proof The set $\mathcal{C}'$ is weakly acyclic, so $chase_{\mathcal{C}'}(\mathcal{A})$ produces a universal solution $\mathcal{A}'$ for $\mathcal{A}$ with respect to $\mathcal{C}'$. We have to show that $\mathcal{A}'$ is also a solution for $\mathcal{A}$ with respect to $\mathcal{C}$, i.e., that also the removed rule, $\sigma = A_n \sqsubseteq \exists R_n.A_1$, holds in $\mathcal{A}'$:

$$\mathcal{A}' \models A_n \sqsubseteq \exists R_n.A_1. \tag{2}$$

Let $\mathcal{A}_1$ be a result of chasing just before applying the rule $\sigma' = \forall x, y((R_1 \circ \cdots \circ R_{n-1})(x, y) \rightarrow R_n(y, x))$. Then

$$\mathcal{A}_1 \models \{A_1(a_1), (R_1 \circ \cdots \circ R_{n-1})(a_1, a_n), A_n(a_n)\},$$

for some constants a_1, a_n. Now, the rule σ' is applicable. After the application:

$$\mathcal{A}_1 \xrightarrow{\sigma', [x \mapsto a_1, y \mapsto a_n]} \mathcal{A}_1 \cup \{R_n(a_n, a_1)\} = \mathcal{A}_2,$$

and $\mathcal{A}_2$ also satisfies σ, i.e., $\mathcal{A}_2 \models A_n \sqsubseteq \exists R_n.A_1$.

The above reasoning shows that every result $\mathcal{A}' = chase_{\mathcal{C}'}(\mathcal{A})$ of chasing that satisfies σ' also satisfies σ. This proves that $\mathcal{A}'$ is a solution for $\mathcal{A}$ with respect to $\mathcal{C}$. However, $\mathcal{A}'$ is not a universal solution for $\mathcal{A}$ with respect to $\mathcal{C}$, that was shown in Example 7. □

The p-aware chase can be generalized to an arbitrary set of RWCs. For $p = (A_1, R_1, A_2, \ldots, A_n, R_n, A_1)$, we denote $\sigma(p) = A_n \sqsubseteq \exists R_n.A_1$.

Definition 14 Let $\mathcal{C}$ be a set of constraint rules, $\mathcal{A}$ a set of facts over $Sig(\mathcal{C}) \cup \mathsf{Const}$, and $\{p_1, \ldots, p_k\}$, $n \geq 1$, a set of RWCs. A $\{p_1, \ldots, p_k\}$-*aware chase* of $\mathcal{A}$ with respect to $\mathcal{C}$, $chase_{\mathcal{C}}^{\{p_1,\ldots,p_n\}}(\mathcal{A})$, is $chase_{\mathcal{C}'}(\mathcal{A})$, such that $\mathcal{C}' = \mathcal{C} \setminus \{\sigma(p_1), \ldots, \sigma(p_k)\}$.

8 Related work

8.1 Ontologies and databases

There is a long research tradition in investigating similarities and differences between ontologies and databases [1,13,14,49,50]. The investigations concern both the underlying theories and behavior in practice. Ontologies offer richer semantics due to the ability to represent both extensional (by means of a set of facts) and intensional (by means of a set of rules) knowledge. Thus, ontologies are commonly considered as the best method to specify conceptualizations of application domains (conceptual models) [11,20,40]. However, the efficiency of query answering on ontologies is unsatisfactory [17,37]. Databases, on the contrary, are characterized by less expressive semantics but higher efficiency. Therefore, databases based on relational or graph models can be used to store ontological instances. Other differences relate to the interpretation of the rules [1,50]. In particular, all rules are interpreted as integrity constraints in databases and as deductive rules in ontologies. To overcome the differences in treating rules in ontologies and in databases, in [50] a concept of the extended knowledge base is proposed, where the set of rules is divided into a set of integrity constraint rules (satisfied in the knowledge base) and a set of deductive rules (representing intensional knowledge). Then, integrity constraints can be disregarded while answering positive queries [50,51]. The combination of ontologies and databases has found a satisfactory solution in ontology-based data access (OBDA) [16,19,63,65,75], where the TBox of an ontology is used as a global schema over a set of integrated databases.

An OBDA specification is a triple $\mathcal{P}' = (\mathcal{O}', \mathcal{M}', Sch)$, where $\mathcal{O}' = (\mathcal{T}, \mathcal{A})$ is an ontology, Sch is a data source schema, and $\mathcal{M}'$ is a mapping from Sch to the signature of $\mathcal{O}'$ [75]. For an instance, $\mathcal{D}$ of Sch, $\mathcal{A} = \mathcal{M}'(\mathcal{D})$, and $\mathcal{M}'$ is a set of source-to-target tuple-generating dependencies of the form: $\forall \mathbf{x}, \mathbf{y}(\Phi(\mathbf{x}, \mathbf{y}) \rightarrow \alpha(\mathbf{x}))$, where Φ is a conjunction of n-ary atoms, and α is an unary or a binary atom. A query q over $\mathcal{O}'$ is rewritten with respect to $\mathcal{T}$ and $\mathcal{M}'$ into a first-order query over the data source. Notice that then: (a) the rules in $\mathcal{T}$ are not satisfied in the data source; instead, they are used in the first-order query rewriting and are not very expressive, usually in *DL-Lite* [17] or sticky [37], (b) integrity constraints are defined and managed in the data source management system.

In this paper, we present a DAFO approach to ontological databases that differs from the OBDA as follows. Let $\mathcal{P} = (\mathcal{O}_{db}, \mathcal{M}, Sch)$ be a DAFO specification, and $\mathcal{P}' = (\mathcal{O}', \mathcal{M}', Sch)$, $\mathcal{O}' = (\mathcal{T}, \mathcal{A})$, an OBDA specification. Then: (a) $\mathcal{O}_{db} = (\mathcal{V}, \mathcal{C}, \mathcal{A}_{db})$, where $\mathcal{T}$ is a subset of $\mathcal{C}$, $\mathcal{T} \subseteq \mathcal{C}$, and is satisfied in $\mathcal{A}_{db}$, as a result of the chase procedure; (b) $\mathcal{M}$ is a mapping from the signature of $\mathcal{A}_{db}$ into Sch consisting of dependencies of the form: $\forall \mathbf{x}(\alpha(\mathbf{x}) \rightarrow \exists \mathbf{y} \mathtt{R}(\mathbf{x}, \mathbf{y}))$, where α ranges over unary and binary atoms, and $\mathtt{R}$ is an n-ary relation name, $n \geq 1$. The inversion of $\mathcal{M}$, in the sense of Fagin [5,32], is an OBDA mapping, (c) $\mathcal{V}$ is a set of rules defining intensional predicates, i.e., predicates not occurring in $\mathcal{A}_{db}$; these predicates are used in query formulation and "compensate" the poor expressive power of mapping dependencies in DAFO; (d) the query rewriting in DAFO is reduced to unfolding intensional predicates with their extensional definitions, and to apply mappings in translating first-order queries into SQL queries.

8.2 Faceted queries over ontologies

In the traditional setting, the reasoning tasks in ontologies include satisfiability and subsumption of concept expressions (with respect to a TBox), and instance checking (with respect to an ABox) [9,61]. New applications of ontology-based systems require not only reasoning capabilities, but also query answering mechanisms [7,16,37,50,69]. Moreover, in [9] it was shown that reasoning tasks over an ontology can be realized by means of queries. Queries over ontologies can be expressed using different query mechanisms [2,7], from first-order logic formulas to graph query languages, such as SPARQL [67], Cypher [36,39,68], or Gremlin [4]. However, such expressive languages as SPARQL are not well-suited for end-users. Thus, we observe attempts to develop interactive graphic-oriented ontology query languages such as, for example, ViziQuer [78], SPARQLGraph [62], OptiqueVQS [66,73], SEWASIE [10], OntoVQL [31], NL-Graphs [27], K-search [12], which present ontology views combined with form-based query entry interfaces. A promising alternative to the aforementioned languages is approaches based on faceted search resulting in faceted queries [7,70].

Faceted search has emerged as a foundation for interactive information browsing and retrieval and has become increasingly prevalent in online information access systems, particularly for e-commerce and site search [7,70–72,74]. Especially significant is combining browsing and searching in more flexible ways to support non-professional end-users in finding information. The implementation of the browsing paradigm allows for exploring and expressing information needs in interactive and iterative ways [42,72,74]. Most importantly, browsing and exploring concerns both data and metadata. Faceted queries are created interactively and iteratively during the faceted search.

The first systems of this kind are /facet [43] and gFacet [42], which identify and implement the basic features of the semantic faceted search paradigm. These two systems operate over RDF data. The expressive power of /facet and gFacet is low. Multiple selections are connected by a logical AND and thus restrict the result set to only objects that satisfy all selections [42]. Exploration is restricted to a navigation during which a conjunction of constraints is added to or removed from a dynamically created faceted query. In gFacet [42], facets and result sets are represented as nodes connected by directed edges labeled by semantic relations between nodes in a graph visualization. More expressive faceted navigation for RDF data was proposed in BrowseRDF [53]. The proposed set of operators describes faceted browsing in terms of a set of manipulations and is defined on an RDF graph. The operators are: basic selection, existential selection, not-existential selection, join and their inversions, and intersection. Further enrichment of the expressive power of exploratory search was proposed in such systems as: Sewelis [35] (allows to search for a limited form of cycles), VisiNav [41] and OpenLink Virtuose [30]. Faceted search solutions are offered as commercial products by some leading software vendors (e.g., ORACLE [52], Microsoft [45], IBM [23] and Apache [3]) There are a large number of implemented systems, which are mostly based on RDF/S. About 30 faceted search systems based on RDF/S datasets are surveyed recently in [72].

Results in [7] can be considered a milestone in the development of the theory of faceted search systems. The authors propose a rigorous theoretical underpinning for faceted search in the context of RDF and OWL 2 ontology profiles. The expressive power of the faceted search language considered in [7] is limited to the description logic positive existential queries (PEQ). The theory is used in the implementation of SemFacet [6,38] and Ontop [15]. Next, the query language in SemFacet has been extended with a restricted form of aggregation and recursion [47,64]. A first approach to view an ontology as a nested facet system for human data integration was proposed in [77].

In this paper, we extend the concept of "flat" facets proposed in [7] to *nested facets*, which are used to propose a *faceted view* of fac-ontologies. A faceted view equipped with a set of operations is defined as a *faceted interface* allowing to explore the ontology and creating queries. The queries are formulated using this graphical tree-shaped faceted interface. This way of querying determines so-called *faceted normal form* (FacNF) of queries. We prove that every expression in $\mathcal{SROIQ}$ (with some limitations) can be converted into FacNF, thus can be created using a faceted interface. This way of formulating queries requires that the ontology meets certain conditions. These conditions are met by fac-ontologies, which are proposed in this paper.

9 Conclusions and future work

In this paper, we have proposed a formal approach and a methodology to create ontological databases with a faceted interface treated as a builder for faceted queries. We identified a class of ontologies, called fac-ontologies, over which a faceted human-oriented interface can be created. We have specified conditions for the class of fac-ontologies and defined the concept of a nested facet, which provides a hierarchical faceted view over fac-ontologies. A hierarchical view with a set of operations constitutes a faceted interface used to formulate queries on the fac-ontology. The set of rules in the TBox consists of two sets: (a) a set $\mathcal{V}$ of rules defining intensional predicates, which extend the vocabulary and enrich the expressive power of the fac-ontology, and (b) a set $\mathcal{C}$ of constraint rules, which are used in the chase procedure to transform a fac-ontology into an ontological database. We show that any query in the description logic $\mathcal{SROIQ}$ (with some restrictions) can be formulated using the proposed faceted interface.

We see many directions for future work. (1) *Transforming heterogeneous data sources into a relational database*. In the DAFO approach, an ontology is mapped to a relational database. In this way, an ontological specification of data sources based on, e.g., XML [29], JSON [26] or RDF [60], can be mapped and materialized in a relational database. Then, the integrated repository can be effectively queried using a faceted interface. (2) *Exploratory search*. Our solution has a rather limited capability to exploratory search of ontologies, which is a characteristic of faceted search systems. Therefore, it would be interesting to enrich the faceted interface with an extended ability to navigate through ontologies with the structure unknown to the user.

The considerations in the paper are based on the DAFO (*Data Access based on Faceted queries over Ontologies*), that was implemented on the top of a commercial relational database engine and ensures high efficiency of query answering. Some details of the implementation as well as the high efficiency of query answering are reported in our previous work [55,56,58].

The performance of DAFO was evaluated on the basis of bibliographic datasets containing data on authors, papers, proceedings, and conferences [56]. The basic dataset was prepared by extracting data from DBLP [1] resources (from XML, HTML, and BibTex files), and enriched with data extracted from personal and conference home pages. This basic dataset includes data on 1907 conferences, 1853 proceedings, 3818

[1] DBLP Computer Science Bibliography, http://dblp.org.

papers, 65 affiliations, and 61 authors. The dataset is organized in the form of an ontological database. The DAFO server is written in C# with NET Core 2.2, DAFO client is written in JavaScript, and the extensional part of the ontological database is stored in SQL Server (under the license Microsoft Imagine Premium). The total execution time consists of the time of: (1) transforming the faceted query into an extensional first-order form, (2) translating the extensional form into an SQL query, (3) executing the SQL query. It turns out that step (1) is the most time-consuming. The experiments showed that the total response time is very promising and for queries similar to the examples in Sect. 6.1 is less than 50 ms. The system is available on GitHub [24].

Acknowledgements This work was supported by the Polish Ministry of Education and Science, grant 0311/SBAD/0710.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley, Reading, MA (1995)
2. Angles, R., Arenas, M., Barceló, P., Hogan, A., Reutter, J.L., Vrgoc, D.: Foundations of modern query languages for graph databases. ACM Comput. Surv. **50**(5), 68:1-68:40 (2017)
3. Apache Solr: https://solr.apache.org/ (2021). Accessed 24 November 2021
4. Apache TinkerPop: http://tinkerpop.apache.org/docs/current/reference/ (2021), Access 24 November 2021
5. Arenas, M., Barceló, P., Libkin, L., Murlak, F.: Relational and XML Data Exchange. Morgan & Claypool Publishers, Synthesis Lectures on Data Management (2010)
6. Arenas, M., Grau, B.C., Kharlamov, E., Marciuska, S., Zheleznyakov, D.: Enabling Faceted Search over OWL 2 with SemFacet. In: OWLED 2014. CEUR, vol. 1265, pp. 121–132 (2014)
7. Arenas, M., Grau, B.C., Kharlamov, E., Marciuska, S., Zheleznyakov, D.: Faceted search over RDF-based knowledge graphs. J. Web Sem. **37–38**, 55–74 (2016)
8. Artale, A., Calvanese, D., Kontchakov, R., Zakharyaschev, M.: The dl-lite family and relations. J. Artif. Intell. Res. **36**, 1–69 (2009)
9. Baader, F., Calvanese, D., McGuinness, D., Nardi, D., Petel-Schneider, P. (eds.): The Description Logic Handbook: Theory, Implementation and Applications. Cambridge University Press, Cambridge (2003)
10. Beneventano, D., Bergamaschi, S., Guerra, F., Vincini, M.: The SEWASIE network of mediator agents for semantic search. J. UCS **13**(12), 1936–1969 (2007)
11. Berardi, D., Calvanese, D., De Giacomo, G.: Reasoning on UML class diagrams. Artif. Intell. **168**, 70–118 (2005)
12. Bhagdev, R., Chapman, S., Ciravegna, F., Lanfranchi, V., Petrelli, D.: Hybrid search: Effectively combining keywords and semantic searches. In: ESWC. pp. 554–568 (2008)
13. Calì, A., Gottlob, G., Lukasiewicz, T., Pieris, A.: Datalog+/-: A family of languages for ontology querying. In: Datalog. LNCS, vol. 6702, pp. 351–368. Springer (2011)
14. Calì, A., Gottlob, G., Pieris, A.: Advanced processing for ontological queries. PVLDB **3**(1), 554–565 (2010)
15. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL queries over relational databases. Semantic Web **8**(3), 471–487 (2017)
16. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Ontology-based database access. In: SEBD 2007. pp. 324–331 (2007)
17. Calvanese, D., Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: the dl-lite family. J. Autom. Reason. **39**(3), 385–429 (2007)
18. Calvanese, D., Giacomo, G.D., Lembo, D., Lenzerini, M., Rosati, R.: EQL-Lite: Effective first-order query processing in description logics. In: IJCAI. pp. 274–279 (2007)
19. Calvanese, D., Horrocks, I., Jiménez-Ruiz, E., Kharlamov, E., Meier, M., Rodriguez-Muro, M., Zheleznyakov, D.: On rewriting, answering queries in OBDA systems for big data. In: OWLED. CEUR, vol. 1080 (2013)
20. Calvanese, D., Lenzerini, M., Nardi, D.: Unifying class-based representation formalisms. J. Artif. Intell. Res. **11**, 199–240 (1999)
21. ten Cate, B., Kolaitis, P.G.: Structural characterizations of schema-mapping languages. Commun. ACM **53**(1), 101–110 (2010)
22. Chen, P.P.: The entity-relationship model - toward a unified view of data. ACM Trans. Database Syst. **1**(1), 9–36 (1976)
23. Creating a faceted enterprise search application: https://www.ibm.com/docs/en/search/faceted?scope=SS5RWK_3.0.0 (2021). Access 24 November 2021
24. DAFO: Data Access based on Faceted queries over Ontology: https://github.com/tpankowski/dafo (2019). Access 24 November 2021
25. Dumais, S.T.: Faceted search. In: Encyclopedia of Database Systems, pp. 1103–1109. Springer (2009)
26. ECMA-404: The JSON data interchange syntax: https://www.ecma-international.org (2017). Access 24 November 2021
27. Elbedweihy, K., Mazumdar, S., Wrigley, S.N., Ciravegna, F.: Nl-graphs: A hybrid approach toward interactively querying semantic data. In: The Semantic Web: Trends and Challenges - ESWC 2014. pp. 565–579 (2014)
28. Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems, 6th edn. Addison-Wesley, Boston (2011)
29. Extensible Markup Language (XML) 1.0 (Fifth Edition): http://www.w3.org/TR/xml/ (2008). Access 24 November 2021
30. Faceted Browsing Tutorial, using LOD Cloud Cache data space: http://vos.openlinksw.com/owiki/wiki/VOS/ (2019). Access 24 November 2021
31. Fadhil, A., Haarslev, V.: OntoVQL: A graphical query language for OWL ontologies. In: DL. CEUR, vol. 250 (2007)
32. Fagin, R.: Inverting schema mappings. ACM Trans. Database Syst. **32**(4), 25:1-25:53 (2007)
33. Fagin, R., Kolaitis, P.G., Miller, R.J., Popa, L.: Data exchange: semantics and query answering. Theor. Comput. Sci. **336**(1), 89–124 (2005)
34. Ferré, S.: Expressive and scalable query-based faceted search over SPARQL endpoints. In: ISWC. LNCS, vol. 8797, pp. 438–453. Springer (2014)
35. Ferré, S., Hermann, A.: Semantic search: Reconciling expressive querying and exploratory search. In: ISWC. LNCS, vol. 7031, pp. 177–192. Springer (2011)

36. Francis, N., Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Rydberg, M., Selmer, P., Taylor, A.: Cypher: An evolving query language for property graphs. In: SIGMOD. pp. 1433–1445. ACM (2018)
37. Gottlob, G., Orsi, G., Pieris, A.: Query rewriting and optimization for ontological databases. ACM Trans. Database Syst. 39(3), 25:1–25:46 (2014)
38. Grau, B.C., Kharlamov, E., Zheleznyakov, D., Arenas, M., Marciuska, S.: On faceted search over knowledge bases. In: DL. CEUR, vol. 1193, pp. 153–156 (2014)
39. Green, A., Guagliardo, P., Libkin, L., Lindaaker, T., Marsault, V., Plantikow, S., Schuster, M., Selmer, P., Voigt, H.: Updating graph databases with cypher. VLDB **12**(12), 2242–2253 (2019)
40. Gruber, T.R.: Toward principles for the design of ontologies used for knowledge sharing? Int. J. Hum.-Comput. Stud. **43**(5–6), 907–928 (1995)
41. Harth, A.: VisiNav: A system for visual search and navigation on web data. J. Web Sem. **8**(4), 348–354 (2010)
42. Heim, P., Ertl, T., Ziegler, J.: Facet Graphs: Complex Semantic Querying Made Easy. In: ESWC. LNCS, vol. 6088, pp. 288–302. Springer (2010)
43. Hildebrand, M., van Ossenbruggen, J., Hardman, L.: /facet: A browser for heterogeneous semantic web repositories. In: ISWC. LNCS, vol. 4273, pp. 272–285. Springer (2006)
44. Horrocks, I., Kutz, O., Sattler, U.: The even more irresistible SROIQ. In: Principles of Knowledge Representation and Reasoning. pp. 57–67. AAAI Press (2006)
45. How to build a facet filter in Azure Cognitive Search: https://docs.microsoft.com/en-us/azure/search/search-filters-facets (2020). Access 24 November 2021
46. Kazakov, Y.: An extension of complex role inclusion axioms in the description logic SROIQ. In: IJCAR. LNCS, vol. 6173, pp. 472–486. Springer (2010)
47. Kharlamov, E., Giacomelli, L., Sherkhonov, E., Grau, B.C., Kostylev, E.V., Horrocks, I.: SemFacet: Making hard faceted search easier. In: CIKM. pp. 2475–2478. ACM (2017)
48. Krötzsch, M., Simancik, F., Horrocks, I.: A description logic primer. CoRR abs/1201.4089 (2012), http://arxiv.org/abs/1201.4089, Access 24 November 2021
49. Libkin, L., Sirangelo, C.: Data exchange and schema mappings in open and closed worlds. J. Comput. Syst. Sci. **77**(3), 542–571 (2011)
50. Motik, B., Horrocks, I., Sattler, U.: Bridging the gap between OWL and relational databases. J. Web Semant. **7**(2), 74–89 (2009)
51. Nikolaou, C., Grau, B.C., Kostylev, E.V., Kaminski, M., Horrocks, I.: Satisfaction and implication of integrity constraints in ontology-based data access. In: IJCAI. pp. 1829–1835 (2019)
52. Oracle Commerce Guided Search: https://docs.oracle.com/cd/E67226_02/Common.112/pdf/GettingStarted.pdf (2015), Access 24 November 2021
53. Oren, E., Delbru, R., Decker, S.: Extending faceted navigation for RDF data. In: ISWC. LNCS, vol. 4273, pp. 559–572. Springer (2006)
54. OWL 2 Web Ontology Language Profiles (Second Edition): (2012), www.w3.org/TR/owl2-profiles, Access 24 November 2021
55. Pankowski, T.: Exploring ontology-enhanced bibliography databases using faceted search. In: TPDL. LNCS, vol. 10450, pp. 27–39. Springer (2017)
56. Pankowski, T.: Rewriting and Executing Faceted Queries over Ontology-Enhanced Databases. In: KES. pp. 137–146. Procedia Computer Science, Elsevier (2017)
57. Pankowski, T., Bak, J.: DAFO: an ontological database system with faceted queries. In: ESWC Satellite Events. LNCS, vol. 11762, pp. 152–155. Springer (2019)
58. Pankowski, T., Brzykcy, G.: Data access based on faceted queries over ontologies. In: DEXA. LNCS, vol. 9828, pp. 275–286. Springer (2016)
59. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Journal on Data Semantics X, pp. 133–173. Springer-Verlag (2008)
60. Resource Description Framework (RDF) Model and Syntax Specification: (1999), www.w3.org/TR/PR-rdf-syntax/, Access 24 November 2021
61. Rudolph, S.: Foundations of description logics. In: Reasoning Web. Semantic Technologies for the Web of Data. LNCS, vol. 6848, pp. 76–136. Springer (2011)
62. Schweiger, D., Trajanoski, Z., Pabinger, S.: SPARQLGraph: a web-based platform for graphically querying biological Semantic Web databases. BMC Bioinformat. **15**(279), 1–55 (2014)
63. Sequeda, J.F., Arenas, M., Miranker, D.P.: OBDA: query rewriting or materialization? In practice, both! In: ISWC. LNCS, vol. 8796, pp. 535–551. Springer (2014)
64. Sherkhonov, E., Grau, B.C., Kharlamov, E., Kostylev, E.V.: Semantic faceted search with aggregation and recursion. In: ISWC. LNCS, vol. 10587, pp. 594–610. Springer (2017)
65. Skjæveland, M.G., Giese, M., Hovland, D., Lian, E.H., Waaler, A.: Engineering ontology-based access to real-world data sources. J. Web Sem. **33**, 112–140 (2015)
66. Soylu, A., Giese, M., Jiménez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I.: Ontology-based end-user visual query formulation: Why, what, who, how, and which? Univ. Access Inf. Soc. **16**(2), 435–467 (2017)
67. SPARQL Query Language for RDF: http://www.w3.org/TR/rdf-sparql-query/ (2008). Access 24 November 2021
68. The Neo4j Cypher Manual v4.1: https://neo4j.com/docs/pdf/neo4j-cypher-manual-4.1.pdf (2020). Access 24 November 2021
69. Thorne, C., Calvanese, D.: Controlled aggregate tree shaped questions over ontologies. In: FQAS. LNCS, vol. 5822, pp. 394–405. Springer (2009)
70. Tunkelang, D.: Faceted Search. Morgan & Claypool Publishers (2009)
71. Tzitzikas, Y., Analyti, A.: Mining the meaningful term conjunctions from materialised faceted taxonomies: algorithms and complexity. Knowl. Inf. Syst. **9**(4), 430–467 (2006)
72. Tzitzikas, Y., Manolis, N., Papadakos, P.: Faceted exploration of RDF/S datasets: a survey. J. Intell. Inf. Syst. **48**(2), 329–364 (2017)
73. Vega-Gorgojo, G., Slaughter, L., Giese, M., Heggestøyl, S., Soylu, A., Waaler, A.: Visual query interfaces for semantic datasets: An evaluation study. J. Web Sem. **39**, 81–96 (2016)
74. Wagner, A., Ladwig, G., Tran, T.: Browsing-oriented semantic faceted search. In: DEXA. LNCS, vol. 6860, pp. 303–319. Springer (2011)
75. Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-based data access: A survey. In: IJCAI. pp. 5511–5519 (2018)
76. Xiao, G., Lanti, D., Kontchakov, R., Komla-Ebri, S., Kalayci, E.G., Ding, L., Corman, J., Cogrel, B., Calvanese, D., Botoeva, E.: The Virtual Knowledge Graph System Ontop. In: ISWC. LNCS, vol. 12507, pp. 259–277. Springer (2020)
77. Zhang, G., Tao, S., Zeng, N., Cui, L.: Ontologies as nested facet systems for human-data interaction. Semantic Web **11**(1), 79–86 (2020)
78. Zviedris, M., Barzdins, G.: ViziQuer: A Tool to Explore and Query SPARQL Endpoints. In: ESWC. LNCS, vol. 6644, pp. 441–445. Springer (2011)