Query Answering Explanation in Inconsistent Datalog+/- Knowledge Bases

Abdallah Arioua, Nouredine Tamani, Madalina Croitoru

Query Answering Explanation in Inconsistent Datalog+/- Knowledge Bases

Abdallah Arioua, Nouredine Tamani, Madalina Croitoru

University Montpellier, France

Abstract. The paper addresses the problem of explaining Boolean Conjunctive Query (BCQ) entailment in the presence of inconsistency within the Ontology-Based Data Access (OBDA) setting, where inconsistency is handled by the intersection of closed repairs semantics (ICR) and the ontology is represented by Datalog+/- rules. We address this problem in the case of both BCQ acceptance and failure by adopting a logical instantiation of abstract argumentation model; that is, in order to explain why the query is accepted or failed, we look for proponent or opponent sets of arguments in favor or against the query acceptance. We have also studied the computational complexity of the problem of finding an arbitrary explanation as well as all explanations.

1 Introduction

In the popular OBDA setting the domain knowledge is represented by an ontology facilitating query answering over existing data [22]. In practical systems involving large amounts of data and multiple data sources, data inconsistency might occur. Many inconsistency-tolerant semantics [7, 6, 17, 18] have been proposed that rely on the notion of data repairs i.e. subsets of maximally consistent data with respect to the ontology. Different query answering strategies over these repairs (called semantics) are investigated in the literature (computing answers that hold in every repair corresponds to AR-semantics, computing answers that hold under the intersection of repairs corresponds to IAR-semantics or to the intersection of closed repairs corresponds to ICR-semantics). Here we only focus on the ICR-semantics. Query answering under these semantics may not be intuitively straightforward and can lead to loss of user's trust, satisfaction and may affect the system's usability [20]. Moreover, as argued by Calvanese et al.[10] *explanation facilities* should not just account for user's "Why Q ?" question (why a query holds under a given inconsistency-tolerant semantics) but also for question like "Why not Q ?" (why a query does not hold under a given inconsistency-tolerant semantics). Given an inconsistent OBDA setting equipped with an inconsistency-tolerant semantics (such as ICR-semantics) and given a boolean conjunctive query Q we consider two query answering problems, namely: ($\mathcal{RQ}_1$) "why does Q hold under such semantics ?" and ($\mathcal{RQ}_2$) "why Q does not hold under such semantics ?". For example, let us a consider a Unique Name Assumption knowledge base where employees work in departments and use offices, some employees direct departments and supervise other employees. Also a supervised employee cannot be a manager. A director of a given department cannot be supervised by an employee of the same department and two employees cannot be direct the same department. We consider factual information stating

that John and Tom direct the finance department, where they both work in. Furthermore Tom is a supervisor of John and it also directs the web department. Let us now consider the query is John an employee. The knowledge base is inconsistent (e. g. Tom and John both direct finance). We can repair it, for instance considering Tom working in finance and directing the web department with John working and directing finance etc. (there are three such repairs). Every repair will entail that John is an employee. However, this is not true of the query asking whether John works in finance because we can consider a repair where this does not hold. These intuitions are formalised in the paper.

In a survey conducted in [25], knowledge base explanation that takes the form of 'Justification' has proven to be more effective than other types of explanation (Line of Reasoning and Strategy for instance). This type of explanation aims at showing the reason why certain conclusions have been drawn in particular circumstances. It is noteworthy that "Justification" is represented as *arguments* in [25]. Following this result, we build upon the logical instantiation of Dung's abstract argumentation framework for OBDA in [11] and exploit their equivalence between ICR-semantics and sceptical acceptance in argumentation framework in order to provide novel explanation facilities for ICR-semantics. In addition, we show how the ICR-semantics can be interpreted in the light of argumentation. We show that ICR-semantics behaves distinctively with respect to the question to be answered. In "Why Q ?" questions, ICR-semantics ensures that a query is accepted if it is supported by a set of arguments that are well-defended. Whereas, in "Why not Q ?" questions, a query fails if its supporting arguments are attacked by a set of arguments that are well-defended. We show the link between our approach and the hitting set based approach of diagnosis [23]. Our work models the hitting set as a set of arguments, as opposed to a set of faulty components. We focus on argumentation because it opens interesting and promising research avenues. The contribution of the paper lies in the following points. First, we propose an explanation for the question "why does Q hold under such semantics ?" based on the set of arguments that support the entailment of the query, called *proponent set* of the query. We define the notion of a *strong proponent set* that will form the basis of the query acceptance explanation. Second, we propose an explanation for the question "why Q does not hold under such semantics ?" based on the set of arguments that does not support the entailment of the query, *opponent set* of the query. Further, we show that these opponent sets play an important role in the failure of the query and form the basis of the query failure explanation. Third, we propose algorithms for finding an arbitrary explanation as well as all explanations. Furthermore, we show the relation between the former problems and the problem of finding minimal hitting sets.

2 Formal Settings and Problem Statement

In this section, we introduce: (1) OBDA setting and representation language, (2) the inconsistency-tolerant semantics and (3) the argumentation framework.

2.1 Language Specification.

There are two major approaches to represent an ontology for the OBDA problem: Description Logics (DL) (such as $\mathcal{EL}$ [4] and DL-Lite [9] families) and rule-based lan-

guages (such as Datalog+/- [8] language, a generalization of Datalog that allows for existentially quantified variables in rules heads). Despite its undecidability when answering conjunctive queries, different decidable fragments of Datalog+/- are studied in the literature [5]. These fragments generalize the above mentioned DL families and overcome their limitations by allowing any predicate arity as well as cyclic structures. Here we restrict ourselves to Datalog+/- classes where the skolemised chase is finite (Finite Expansion Sets). We consider *the positive existential* conjunctive fragment of first-order logic, denoted by FOL($\wedge$,$\exists$), which is composed of formulas built with the connectors $(\wedge, \rightarrow)$ and the quantifiers $(\exists, \forall)$. We consider first-order vocabularies with constants but no other function symbol. A term t is a constant or a variable, different constants represent different values (unique name assumption), an atomic formula (or atom) is of the form $p(t_1, ..., t_n)$ where p is an n-ary predicate, and $t_1, ..., t_n$ are terms. A *ground* atom is an atom with no variables. A variable in a formula is free if it is not in the scope of a quantifier. A formula is closed if it has not free variable. We denote by $\mathbf{X}$ (with a bold font) sequences of variables $X_1, .., X_k$ with $k \geq 1$. A *conjunct* $C[\mathbf{X}]$ is a finite conjunction of atoms, where $\mathbf{X}$ is the sequence of variables occurring in C. Given an atom or a set of atoms A, $vars(A)$, $consts(A)$ and $terms(A)$ denote its set of variables, constants and terms, respectively.

An *existential rule* (rule) is a first-order formula of the form $r = \forall \mathbf{X} \forall \mathbf{Y}(H[\mathbf{X}, \mathbf{Y}]) \rightarrow \exists \mathbf{Z} C[\mathbf{Z}, \mathbf{Y}]$, with $vars(H) = \mathbf{X} \cup \mathbf{Y}$, and $vars(C) = \mathbf{Z} \cup \mathbf{Y}$ where H and C are *conjuncts* called the hypothesis and conclusion of R, respectively. We denote by $R = (H, C)$ a contracted form of a rule R. An existential rule with an empty hypothesis is called a *fact*. A fact is an existentially closed (with no free variable) conjunct.

We recall that a *homomorphism* π from a set of atoms A_1 to set of atoms A_2 is a substitution of $vars(A_1)$ by $terms(A_2)$ such that $\pi(A_1) \subseteq A_2$. Given two facts f and f' we have $f \models f'$ iff there is a homomorphism from f' to f, where $\models$ is the first-order semantic entailment. A *rule* $r = (H, C)$ is *applicable* to a set of facts F iff there exists $F' \subseteq F$ such that there is a homomorphism π from H to the conjunction of elements of F'. If a rule r is applicable to a set F, its application according to π produces a set $F \cup \{\pi(C)\}$. The new set $F \cup \{\pi(C)\}$, denoted also by $r(F)$, is called *immediate derivation* of F by r. A *negative constraint* is a first-order formula $n = \forall \mathbf{X}\, H[\mathbf{X}] \rightarrow \bot$ where $H[\mathbf{X}]$ is a conjunct called hypothesis of n and $\mathbf{X}$ sequence of variables appearing in the hypothesis. **Knowledge base.** A knowledge base $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ is composed of finite set of facts $\mathcal{F}$ and finite set of existential rules $\mathcal{R}$ and a finite set of negative constrains $\mathcal{N}$.

Example 1. The following example is inspired by [8]. In an enterprise, employees work in departments and use offices, some employees direct departments, and supervise other employees. In addition, a supervised employee cannot be a manager. A director of a given department cannot be supervised by an employee of the same department, and two employees cannot direct the same department. The following sets of (existential) rules $\mathcal{R}$ and negative constraints $\mathcal{N}$ model the corresponding ontology:

$$\mathcal{R} = \begin{cases} \forall x \forall y\, (works\_in(x, y) \rightarrow emp(x)) & (r_1) \\ \forall x \forall y\, (directs(x, y) \rightarrow emp(x)) & (r_2) \\ \forall x \forall y\, (directs(x, y) \wedge works\_in(x, y) \rightarrow manager(x)) & (r_3) \\ \forall x\, (emp(x) \rightarrow \exists y(office(y) \wedge uses(x, y))) & (r_4) \end{cases}$$

$$\mathcal{N} = \begin{cases} \forall x \forall y\, (supervises(x,y) \wedge manager(y)) \rightarrow \bot & (n_1) \\ \forall x \forall y \forall z\, (supervises(x,y) \wedge works\_in(x,z) \wedge directs(y,z) \wedge x \neq y) \rightarrow \bot & (n_2) \\ \forall x \forall y \forall z (directs(x,z) \wedge directs(y,z) \wedge x \neq y) \rightarrow \bot & (n_3) \end{cases}$$

Let us suppose the following set of facts $\mathcal{F}$ that represent explicit knowledge:

$$\mathcal{F} = \begin{cases} directs(John, d_1)\ (f_1) & directs(Tom, d_1)\ (f_2) \\ directs(Tom, d_2)\ (f_3) & supervises(Tom, John)\ (f_4) \\ works\_in(John, d_1)\ (f_5) & works\_in(Tom, d_1)\ (f_6) \end{cases}$$

$\mathcal{R}$-derivation. Let $F \subseteq \mathcal{F}$ be a set of facts and $\mathcal{R}$ be a set of rules. An $\mathcal{R}$-derivation of F in $\mathcal{K}$ is a finite sequence $\langle F_0, ..., F_n \rangle$ of sets of facts s.t $F_0 = F$, and for all $i \in \{0, ..., n\}$ there is a rule $r_i = (H_i, C_i) \in \mathcal{R}$ and a homomorphism π_i from H_i to F_i s.t $F_{i+1} = F_i \cup \{\pi(C_i)\}$. For a set of facts $F \subseteq \mathcal{F}$ and a query Q and a set of rules $\mathcal{R}$, we say $F, \mathcal{R} \models Q$ iff there exists an $\mathcal{R}$-derivation $\langle F_0, ..., F_n \rangle$ such that $F_n \models Q$.

Closure. Given a set of facts $F \subseteq \mathcal{F}$ and a set of rules $\mathcal{R}$, the closure of F with respect to $\mathcal{R}$, denoted by $\mathtt{Cl}_{\mathcal{R}}(F)$, is defined as the smallest set (with respect to $\subseteq$) which contains F and is closed under $\mathcal{R}$-derivation. By considering the skolemised chase and the finite fragments of Datalog+/- this set is unique (i.e. universal model).

Finally, we say that a set of facts $F \subseteq \mathcal{F}$ and a set of rules $\mathcal{R}$ *entail* a fact f (and we write $F, \mathcal{R} \models f$) iff the closure of F by all the rules entails f (i.e. $\mathtt{Cl}_{\mathcal{R}}(F) \models f$).

A conjunctive query (CQ) has the form $Q(\mathbf{X}) = \exists \mathbf{Y} \Phi[\mathbf{X}, \mathbf{Y}]$ where $\Phi[\mathbf{X}, \mathbf{Y}]$ is a conjunct such that $\mathbf{X}$ and $\mathbf{Y}$ are variables appearing in Φ. A Boolean CQ (BCQ) is a CQ of the form $Q()$ with an answer yes or no. We refer to a BCQ with the answer *no* as **failed query**, whereas a query with the answer *yes* as **accepted query**.

Inconsistency-Tolerant Semantics. Given a knowledge base $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$, a set $F \subseteq \mathcal{F}$ is said to be *inconsistent* iff there exists a constraint $n \in \mathcal{N}$ such that $F \models H_n$, where H_n is the hypothesis of the constraint n. A set of facts is consistent iff it is not inconsistent. A set $F \subseteq \mathcal{F}$ is *$\mathcal{R}$-inconsistent* iff there exists a constraint $n \in \mathcal{N}$ such that $\mathtt{Cl}_{\mathcal{R}}(F) \models H_n$. A set of facts is said to be *$\mathcal{R}$-inconsistent* iff it is not *$\mathcal{R}$-consistent*. A knowledge base $(\mathcal{F}, \mathcal{R}, \mathcal{N})$ is said to be *inconsistent* iff $\mathcal{F}$ is $\mathcal{R}$-inconsistent.

Notice that (like in classical logic), if a knowledge base $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ is inconsistent, then everything is entailed from it. A common solution [6, 17] is to construct maximal (with respect to set inclusion) consistent subsets of $\mathcal{K}$. In this finite chase case there is a finite number of such sets. They are called *repairs* and denoted by $\mathcal{R}epair(\mathcal{K})$. Once the repairs are computed, different semantics can be used for query answering over the knowledge base. In this paper we focus on (**I**ntersection of **C**losed **R**epairs semantics) [6].

Definition 1 (ICR-semantics). *Let $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ be a knowledge base and let Q be a query. Q is ICR-entailed from $\mathcal{K}$, written $\mathcal{K} \models_{ICR} Q$ iff $\bigcap_{\mathcal{A} \in \mathcal{R}epair(\mathcal{K})} \mathtt{Cl}_{\mathcal{R}}(\mathcal{A}) \models Q$.*

Example 2. The knowledge base in Example 1 is inconsistent because the set of facts $\{f_1, f_4, f_6\} \subseteq \mathcal{F}$ is inconsistent since it violates the negative constraint n_2. In this example we obtain 3 repairs. The following is one of them:

$\mathcal{A}_1 = \{directs(John, d_1), supervises(Tom, John), directs(Tom, d_2)\}$

The closure of $\mathcal{A}_1$ is:

$\mathtt{Cl}_{\mathcal{R}}(\mathcal{A}_1) = \{directs(John, d_1), supervises(Tom, John),$
$directs(Tom, d_2), manager(Tom), emp(John), emp(Tom),$
$\exists y_1(\mathit{office}(y_1) \wedge uses(Tom, y_1))),$
$\exists y_2(\mathit{office}(y_2) \wedge uses(John, y_2)))\}$

2.2 Problem Statement

In what follows we introduce the motivation of providing explanation facilities for Inconsistent OBDA.

Example 3. Consider the query $Q = emp(John)$ in the knowledge base of Example 1. Q is accepted under ICR-semantics since it follows from the intersection of all closed repairs. The user may be interested in knowing why this is the case.
The query $Q' = works\_in(John, d_1)$ is a failed query since it does not follow from the intersection of all closed repairs. Such query failure may need an explanation when the user is expecting the acceptance of the query.

The above example shows the need for explanation facilities with the aim of helping the user to understand why a query holds or fails. Thus, we define our problem as: given an inconsistent knowledge base $\mathcal{K}$ and a boolean conjunctive query Q:

$\mathcal{RQ}_1$**:** why Q does hold under the ICR-semantics in the knowledge base $\mathcal{K}$?
$\mathcal{RQ}_2$**:** why Q' does not hold under the ICR-semantics in the knowledge base $\mathcal{K}$?

The Proposed Solution. To depict the intuition underlying our proposal, in what follows we show how an explanation can be provided by means of argumentation within the context of an inconsistent OBDA.

Example 4 (Explanation). Let us consider the query $Q = emp(John)$. The knowledge base of Example 1 is inconsistent thus we have the following consistent possibilities:

1. Tom works in d_1 and directs d_2, while John works in d_1 and directs d_1.
2. Tom directs d_2 and supervises John who directs d_1.
3. Tom works in d_1, directs d_1 and d_2, and supervises John who in turn works in d_1.

Please note that the depiction of logical rules in textual form is for illustration purposes only. Thus, in this paper we do not address the problem of generating textual explanation based on natural language processing.

From above we can always infer $Q = emp(John)$ since from (1) and (3) we can construct (including and not limited to) the argument "John is an employee because he works in d_1", and from (2) we get "John is an employee because he directs d_1". In other words, the query Q is supported from all possible "points of view". This support is in a form of arguments, these arguments are grouped in what we call a "proponent set".

As for query $Q' = works\_in(John, d_1)$, we can see that in (1) and (3) we have an argument that says "John works in d_1" which supports Q'. However, the query is not supported from all points of view, for instance in (2) we have the argument "John directs d_1 and he is supervised by Tom, so John cannot work in d_1 since he would be a manager and a manager cannot be supervised" that attacks the argument "John works in d_1'. It seems that the former argument is holding the query Q' from being accepted from all possible points of view. Note that, there may be more than one argument that jointly hold the query from being accepted. These arguments are grouped in what we call "opponent set".

In the two cases (Q and Q') we consider proponent and opponent sets as the core of our explanation. Moreover, we deepen the explanation by considering possible attacks

scenario between arguments aiming at showing the user the dialectical procedure that allows the acceptance or the failure of the query. To do so, we make use of Dung's argumentation frameworks [12] adapted for the aforementioned rule-based language in [11]. First let us specify the structure of an argument under such language.

2.3 Rule-Based Dung Argumentation Framework Instantiation

As defined in [11], given a knowledge base $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$, the *corresponding argumentation framework* $\mathcal{AF}_{\mathcal{K}}$ is a pair $(\texttt{Arg}, \texttt{Att})$ where $\texttt{Arg}$ is the set of arguments that can be constructed from $\mathcal{F}$ and $\texttt{Att}$ is an asymmetric binary relation called *attack* defined over $\texttt{Arg} \times \texttt{Arg}$. An argument is defined as follows.

Definition 2 (Argument [11]). *Given a knowledge base $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$, an argument a is a tuple $a = (F_0, F_1, \ldots, F_n, C)$ where:*

- *$(F_0, \ldots, F_n)$ is an $\mathcal{R}$-derivation of F_0 in $\mathcal{K}$,*
- *C is an atom, a conjunction of atoms, the existential closure of an atom or the existential closure of a conjunction of atoms such that $F_n \models C$.*

F_0 is the support of the argument a (denoted $\texttt{Supp}(a)$) and C is its conclusion (denoted $\texttt{Conc}(a)$).

Example 5 (Argument). The following argument indicates that John is an employee because he directs department d_1:

$a = (\{directs(John, d_1)\}, \{directs(John, d_1), emp(John)\}, emp(John))$.

Definition 3 (Attack [11]). *The attack between two arguments expresses the conflict between their conclusion and support. We say that an argument a attacks an argument b iff there exists a fact $f \in \texttt{Supp}(a)$ such that the set $\{\texttt{Conc}(b), f\}$ is $\mathcal{R}$-inconsistent.*

Example 6 (Attack). Consider the argument a of Example 5, the following argument $b = (\{supervises(Tom, John), works\_in(Tom, d_1)\}, supervises(Tom, John) \wedge works\_in(Tom, d_1))$ attacks a, because $\{supervises(Tom, John) \wedge works\_in(Tom, d_1), directs(John, d_1)\}$ is $\mathcal{R}$-inconsistent since it violates the constraint n_2.

Admissibility, Semantics and Extensions in $\mathcal{AF}$. Let $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ be a knowledge base and $\mathcal{AF}_{\mathcal{K}}$ its *corresponding argumentation framework*. Let $\mathcal{E} \subseteq \texttt{Arg}$ be a set of arguments. We say that $\mathcal{E}$ is *conflict free* iff there exist no arguments $a, b \in \mathcal{E}$ such that $(a, b) \in \texttt{Att}$. $\mathcal{E}$ *defends* an argument a iff for every argument $b \in \texttt{Arg}$, if we have $(b, a) \in \texttt{Att}$ then there exists $c \in \mathcal{E}$ such that $(c, b) \in \texttt{Att}$. $\mathcal{E}$ is *admissible* iff it is conflict free and defends all its arguments. $\mathcal{E}$ is a *preferred extension* iff it is maximal (with respect to set inclusion) admissible set (please see [12] for other types of semantics).

We denote by $\texttt{Ext}(\mathcal{AF}_{\mathcal{K}})$ the set of all extensions of $\mathcal{AF}_{\mathcal{K}}$. An argument is sceptically accepted if it is in all extensions, credulously accepted if it is in at least one extension and not accepted if it is not in any extension.

Equivalence between ICR and Preferred semantics. Let $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ be a knowledge base and $\mathcal{AF}_{\mathcal{K}}$ the corresponding argumentation framework. A query Q is sceptically accepted under preferred semantics iff $(\bigcap_{\mathcal{E} \in \texttt{Ext}(\mathcal{AF}_{\mathcal{K}})} \texttt{Concs}(\mathcal{E})) \models Q$, such that $\texttt{Concs}(\mathcal{E}) = \bigcup_{a \in \mathcal{E}} \texttt{Conc}(a)$. The results of [11] show the equivalence between sceptically acceptance under preferred semantics and ICR-entailment:

Theorem 1 (Semantics equivalence[11]). *Let $\mathcal{K} = (\mathcal{F}, \mathcal{R}, \mathcal{N})$ be a knowledge base, let $\mathcal{AF}_{\mathcal{K}}$ the corresponding argumentation framework, Q a query. $\mathcal{K} \models_{ICR} Q$ iff Q sceptically accepted under preferred semantics.*

3 Argumentative Explanation

In this section we introduce the argumentation-based explanation for query acceptance (Subsection 3.1) and failure (Subsection 3.2). Note that due to the finiteness property of our language the argumentative framework is finite.

3.1 Explaining Query Acceptance

We introduce the notion of supporting argument of a query, notion needed when selecting the arguments for or against a query:

Definition 4 (Supporting argument). *Let $\mathcal{K}$ be a knowledge base, $\mathcal{AF}_{\mathcal{K}} = (\texttt{Arg}, \texttt{Att})$ the corresponding argumentation framework, Q a query. An argument $a \in \texttt{Arg}$ supports Q iff $\texttt{Conc}(a) \vdash Q$. The set of all supporting arguments of Q is denoted by $\mathcal{S}up(Q)$.*

A query may have more than one supporting argument with different statuses (accepted or not accepted). The arguments with an accepted status (belong to at least one extension) form a *proponent set* of the query Q. We exclude not accepted arguments since they cannot form a base for the support of the acceptance of the query.

Definition 5 (Proponent set). *Let $\mathcal{K}$ be a knowledge base, $\mathcal{AF}_{\mathcal{K}}$ the corresponding argumentation framework, $\texttt{Ext}(\mathcal{AF}_{\mathcal{K}})$ set of all preferred extensions of $\mathcal{AF}_{\mathcal{K}}$, Q a query and $\mathcal{S}up(Q)$ the set of all supporting arguments of Q. Then, the set $\mathcal{P} \subseteq \mathcal{S}up(Q)$ is a proponent set of Q iff: (1) $\forall a \in \mathcal{P}$ there exists an extension $\mathcal{E} \in \texttt{Ext}(\mathcal{AF}_{\mathcal{K}})$ such that $a \in \mathcal{E}$; (2) there exists no $\mathcal{P}'$ such that $\mathcal{P} \subset \mathcal{P}'$ and $\mathcal{P}'$ satisfies (1).*

Note that there is only one proponent set of any query Q.

Example 7. Consider a knowledge base $\mathcal{K}$ that corresponds to an argumentation framework s.t. the set of arguments is $\texttt{Arg} = \{a, b, c, d, e, f, g, h\}$ and the attack relation is: $\texttt{Att} = \{(f, a), (c, f), (e, c), (b, e), (b, f), (d, b), (c, d), (c, g), (h, g)\}$. For space reasons the knowledge base underpinning this argumentation framework is not shown. Let Q be a query. Suppose that we have $\texttt{Conc}(a) \models Q$, $\texttt{Conc}(d) \models Q$ and $\texttt{Conc}(g) \models Q$. This means that the set of supporting arguments of Q is $\{a, d, g\}$. The set of preferred extensions is $\texttt{Ext}(\mathcal{AF}_{\mathcal{K}}) = \{\mathcal{E}_1 = \{a, c, b, h\}, \mathcal{E}_2 = \{e, f, d, h\}\}$. The proponent set $\mathcal{P}$ of Q is $\{a, d\}$. Notice that $g \notin \mathcal{P}$ since it does not belong to any extension.

A stronger notion of support for a query is expressed in what we call strong proponent set. The arguments in this set are forced to be distributed over all extensions.

Definition 6 (Strong proponent set). *Let $\mathcal{K}$ be a knowledge base, $\mathcal{AF}_{\mathcal{K}}$ the corresponding argumentation framework, $\texttt{Ext}(\mathcal{AF}_{\mathcal{K}})$ set of all preferred extensions of $\mathcal{AF}_{\mathcal{K}}$, Q a query and $\mathcal{P}$ the proponent set of Q. Then, a set $\mathcal{S} \subseteq \mathcal{P}$ is a strong proponent set of Q iff: (1) $\forall \mathcal{E} \in \texttt{Ext}(\mathcal{AF}_{\mathcal{K}})$ there exists an argument $a \in \mathcal{S}$ such that $a \in \mathcal{E}$. (2)*

there exists no proper subset $\mathcal{S}' \subset \mathcal{S}$ such that $\mathcal{S}'$ satisfies (1). We say the proponent set $\mathcal{P}$ is strong if it has at least one strong proponent set. The set of all strong proponent sets of a query Q is denoted by $\mathcal{S}trong(Q)$. Computing $\mathcal{S}trong(Q)$ is called the strong proponent set problem.

Example 8. Consider the same argumentation framework of Example 7 and make the slight modification such that the set of supporting arguments of Q is $\{a, g\}$. In this case, the proponent set of Q is $\mathcal{P} = \{a\}$. Note that Q has no strong proponent set since there is an extension $\mathcal{E}_2 = \{e, f, d, h\}$ for which we can never construct a set $\mathcal{S} \subseteq \mathcal{P}$ in which there is an argument that belongs to $\mathcal{E}_2$.

If a query has a strong proponent set then this indicates that the query is supported from all possible extensions. This property has a strong relation with ICR-semantics as expressed by the next proposition.

Proposition 1. *Let $\mathcal{K}$ be a knowledge base, $\mathcal{AF}_{\mathcal{K}}$ the corresponding argumentation framework, Q a query and $\mathcal{P}$ its proponent set. Then $\mathcal{K} \models_{ICR} Q$ iff $\mathcal{P}$ is strong.*

Proof. In the technical report [1] Page 7.

Proposition 1 indicates that the strong proponent set represents the reason why a query is accepted under ICR-semantics. The ICR-semantics, in fact, operates such that if the query is not supported from one extension then it should not be accepted. So, strong proponent set will form the basis of an explanation for query acceptance under ICR-semantics.

Example 9 (Count. Example 7). In order to provide an explanation for the acceptance of the query Q in Example 7, we should mention the fact that the query is supported by the well-defended arguments a and d. In general, we need to show that whenever an argument is attacked, there is an argument or a set of arguments that defend it. For instance, the argument a is attacked by f but it is defended by c and b. Moreover, the argument d is attacked by c but it is defended by e.

The intuition captured in the previous example is formalized as a tree structure called a *defense tree*.

Definition 7 (Defense Tree). *Let $a \in$ `Arg`. A Defense Tree $\mathcal{DT}(a)$ of a is a tree of arguments such that: (1) $H(\mathcal{DT}(a)) = 3$; (2) $\mathcal{DT}(a)$'s root node is the argument a, and $\forall b, c \in$ `Arg` : b is a child of c in $\mathcal{DT}(a)$ iff $(b, c) \in$ `Att`; (3) $\{a \cup leafs(\mathcal{DT}(a))\}$ is conflict-free; (4) every internal node has a child. Notice that $H(\mathcal{DT}(a))$ indicates the height of the tree and $leafs(\mathcal{DT}(a))$ indicates the leafs of the tree.*

Example 10. Figure 1 depicts a defense tree for the arguments a and c of Example 7.

Note that, if a query Q is accepted, then all arguments in $\mathcal{S}trong(Q)$ have a defense tree since they are accepted with respect to an extension. Given a set of arguments S, the set of all defense trees for all arguments in S is denoted $\mathcal{DT}(S)$ and defined as $\mathcal{DT}(S) = \{\mathcal{DT}(a) | a \in S\}$.

Based on this notion, an explanation in our framework is not just the strong proponent set, but also a description of how these arguments are defended.

Fig. 1: Defense trees of the arguments a (left) and d (right).

Definition 8 (Explanation for query acceptance). *Let $\mathcal{K}$ be an inconsistent knowledge base, Q a query such that $\mathcal{K} \models_{ICR} Q$. An explanation $E^{+}_{\mathcal{K}}(Q)$ for the acceptance of Q in $\mathcal{K}$ is the tuple $\langle \mathcal{S}, \mathcal{DT}(\mathcal{S}) \rangle$ such that $\mathcal{S}$ is a strong proponent set of Q and $\mathcal{DT}(\mathcal{S})$ is the set of all defense trees of arguments in $\mathcal{S}$. The set of all explanations for the acceptance of Q is denoted by $\texttt{exp}^{+}(Q)$.*

3.2 Explaining Query Failure

In this subsection we consider the problem of explaining why a query Q is not ICR-entailed. In this case the failure of a query is more related to "the opposition against the query" than " the support for the query". Let us start by defining an opponent set.

Definition 9 (Opponent set). *Let $\mathcal{K}$ be an inconsistent knowledge base and $\mathcal{AF}_{\mathcal{K}} = (\texttt{Arg}, \texttt{Att})$ its corresponding argumentation framework, Q be a query and let $\mathcal{P}$ be a proponent set of Q. A finite set of arguments $\mathcal{O}$ is an opponent set of Q iff:*

1. *$\forall a \in \mathcal{O}$ there exists an extension $\mathcal{E} \in \texttt{Ext}(\mathcal{AF}_{\mathcal{K}})$ such that $a \in \mathcal{E}$.*
2. *For every $b \in \mathcal{P}$, there exists $a \in \mathcal{O}$ such that $(a, b) \in \texttt{Att}$.*
3. *$\forall a \in \mathcal{O}$ and $\forall b \in \mathcal{P}$, it never holds that a and b are in the same extension.*
4. *for all $a \in \mathcal{O}$ there exists an argument $b \in \mathcal{P}$ such that $(a, b) \in \texttt{Att}$.*

An opponent set is defined with respect to a proponent set since an opponent of Q is a set of arguments that are against the arguments that support Q.

As for property (1) and (2) represent minimal requirements, i.e. acceptance and attack. Property (3) insures coherence, that means an argument that belongs to an extension should not be against a query Q while there are some arguments in its extension which are supporting Q. The last property reinforces the property (2) and eliminates redundancy. This means that an opponent set has to incorporate only arguments that participate in the attack on the proponent set. Notice that, according to the definition above, there may be more than one opponent set of a query. Thus, the set of all opponent sets of a query Q is denoted by $\mathcal{OPP}(Q)$.

Example 11 (Opponent of a query). Consider an argumentation framework $\mathcal{AF}_{\mathcal{K}}$ s.t. $\texttt{Ext}(\mathcal{AF}_{\mathcal{K}}) = \{\mathcal{E}_1 = \{a, b, c\}, \mathcal{E}_2 = \{d, e, f\}, \mathcal{E}_3 = \{g, h, k, m\}\}$ and its attack relation contains the following attacks among others $\{(f, a), (h, c)\}$. Consider a query Q with a proponent set $\mathcal{P} = \{a, c\}$. Then, the set of all opponent sets is $\mathcal{OPP}(Q) =$

$\{\{f, h\}\}$. As stated in Definition 9 all the arguments of $\mathcal{P}$ are attacked (the argument f (resp. h) attacks a (resp. c)). In addition, the argument f is accepted w.r.t the extension $\mathcal{E}_2$ and the argument h is accepted w.r.t the extension $\mathcal{E}_3$. The presence of the argument f (resp. h) in $\mathcal{E}_2$ (resp. $\mathcal{E}_3$) prevents the query to be inferred from all extensions.

The relation between opponent set and the acceptance and failure of a query is presented i what follows.

Proposition 2. *Let $\mathcal{K}$ be an inconsistent knowledge base, Q be a query and let $\mathcal{P}$ be a proponent set of Q. Then, $\mathcal{P}$ is strong iff $\mathcal{OPP}(Q) = \emptyset$.*

Corollary 1. *Let $\mathcal{K}$ be an inconsistent knowledge base, Q be a query and let $\mathcal{P}$ be a non-empty proponent set of Q. Then, $\mathcal{K} \models_{ICR} Q$ iff $\mathcal{OPP}(Q) = \emptyset$.*

Proof. In [1] pages 9-10.

Proposition 2 and Corollary 1 indicate that if the query has a support represented by the proponent set $\mathcal{P}$, and an opposition represented by opponent sets, then the query is not accepted. In this case, an opponent set of a query Q plays an important role in the failure of the query and can form the basis of the explanation.

Like in explaining query acceptance, we do not limit the explanation to the arguments that force the query to fail, we also present how these arguments are defended.

Definition 10 (Explanation for query failure). *Let $\mathcal{K}$ be an inconsistent knowledge base, Q a query such that $\mathcal{K} \not\models_{ICR} Q$. An explanation $E^{-}_{\mathcal{K}}(Q)$ for the failure of Q in $\mathcal{K}$ is the tuple $\langle \mathcal{O}, \mathcal{DT}(\mathcal{O}) \rangle$ such that $\mathcal{O}$ is an opponent set for Q and $\mathcal{DT}(\mathcal{S})$ is the set of all defense trees of arguments in S. The set $\mathtt{exp}^{-}(Q)$ is the set of all $E^{-}_{\mathcal{K}}(Q)$.*

The selection of the set $\mathcal{O}$ (resp. $\mathcal{S}$ in Definition 8) is arbitrary and it can be defined according to some criteria, which their specification is beyond the scope of the paper.

4 Algorithms

As mentioned in Section 2, boolean conjunctive query answering in Datalog+/- under general existential rules is undecidable. Thus, the results and the algorithms presented in this section are considered only on families of Datalog+/- for which BCQ answering is tractable w.r.t data complexity[1](for a detailed review see [8, 14]). In addition, in these fragments the *closure* ($\mathtt{Cl}_{\mathcal{R}}(\mathcal{F})$) over any set of facts $\mathcal{F}$ is finite and any $\mathcal{R}$-derivation from a knowledge base is finite (see [8, 14]). Consequently, the corresponding argumentation framework $\mathcal{AF}_{\mathcal{K}} = (\mathtt{Arg}(\mathcal{F}), \mathtt{Att})$ of the knowledge base $\mathcal{K}$ is finite. Precisely, the set of all arguments $\mathtt{Arg}(\mathcal{F})$ and the set of all extensions $\mathtt{Ext}(\mathcal{AF}_{\mathcal{K}})$ are finite. Furthermore, since an argument is an $\mathcal{R}$-derivation and an $\mathcal{R}$-derivation is finite so all the arguments of $\mathcal{AF}_{\mathcal{K}}$ are finite.

An explanation has two components: a defense tree and strong proponent set (resp. opponent set). Since the defense tree is a common component for query acceptance and failure explanation, we start in the next subsection by presenting Algorithm 1 that

[1] This refers to the complexity of evaluating the query over the knowledge base where the size of the query is assumed to be constant.

Algorithm 1 DEFENSETREE

```
1: function DefenseTree(a, Ext(AF_K))
2:     DefTree = ∅; Attackers = ∅; Defenders = ∅
3:     for all E ∈ Ext(AF_K) such that a ∉ E do
4:         Attackers = Attackers ∪ {b | b ∈ E such that b attacks a}
5:         DefTree = DefTree ∪ {(b, a) | b ∈ Attackers}
6:     for all E ∈ Ext(AF_K) do
7:         for all b ∈ Attackers do
8:             Defenders = Defenders ∪ {c | c ∈ E such that c attacks b}
9:             DefTree = DefTree ∪ {(c, b) | c ∈ Defenders}
10:    Return (DefTree)
```

computes a defense tree for a given argument. Then, before presenting algorithms for computing strong proponent set and opponent set, we show the relation between the former sets (separately) and minimal hitting sets and present Algorithm 2 & 3 that establish such relation. Next, in Algorithm 4 we present the algorithm that compute minimal hitting set in order to use it in the final algorithm (Algorithm 5) to compute an explanation.

4.1 Computing Defense Tree

The function *DefenseTree()* in Algorithm 1 takes as an input an argument a and a set of extensions. First, it computes the set of attackers of a (for-loop at line 4) then for each attacker it computes its attackers (for-loop at line 7). Finally, the function returns the defense tree as a form of a binary relation called *DefTree*. The time complexity of the algorithm is mainly related to the computation of attack between two arguments in line 5 and 9 (since all the loops induce only a polynomial overhead). This issue comes down to check if the conclusion and the hypothesis of the two arguments are unsatisfiable in $\mathcal{K}$ which can be done efficiently ([8]).

4.2 Computing Strong Proponent and Opponent sets

In what follows we define the Hitting Set Problem.

Definition 11 (Hitting Set Problem). *Given a collection $\mathcal{C} = \{S_1, ..., S_m\}$ of finite nonempty subsets of a set $\mathcal{B}$ (the background set). A* hitting set *of $\mathcal{C}$ is a set $\mathcal{H} \subseteq \mathcal{B}$ such that $S_j \cap \mathcal{H} \neq \emptyset$ for all $S_j \in \mathcal{C}$. A hitting set for $\mathcal{C}$ is minimal iff no proper subset of it is a hitting set for $\mathcal{C}$. The set of all minimal hitting sets is denoted by $\mathcal{M}$. Computing $\mathcal{M}$ is known as the hitting set problem.*

The hitting set problem amounts to find all minimal (w.r.t $\subseteq$) sets of $\mathcal{B}$ whose intersection with each set of $\mathcal{C}$ is non-empty.

Reducibility. Recall that, the problem of computing strong proponent sets (resp. opponent sets) is defined as the problem of computing all strong proponent sets (resp. opponent sets) for a given query Q. It turns out that the problem of computing strong proponent sets (resp. opponent sets) is polynomially reducible to the hitting set problem. As its name indicates, Algorithm 2 (resp. Algorithm 3) is the algorithm responsible for

Algorithm 2 REDUCESTRONGTOHITTING

function REDUCESTRONGTOHITTING($\texttt{Ext}(\mathcal{AF}_\mathcal{K})$,$Q$)
  $\mathcal{B} = \emptyset; \mathcal{C} = \{\{\}\}$
  for all $\mathcal{E} \in \texttt{Ext}(\mathcal{AF}_\mathcal{K})$ **do**
    $S = \{a | a \in \mathcal{E}$ *such that* $\texttt{Conc}(a) \models Q\}$
    $\mathcal{C} = \mathcal{C} \cup \{S\}$
  $\mathcal{B} = \bigcup_{S \in \mathcal{C}} S$
  Return $(\mathcal{C}, \mathcal{B})$

Algorithm 3 REDUCEOPPONENTTOHITTING

function REDUCEOPPONENTTOHITTING($\texttt{Ext}(\mathcal{AF}_\mathcal{K})$,$Q$)
  $\mathcal{B} = \emptyset; \mathcal{C} = \{\{\}\}$
  for all $\mathcal{E} \in \texttt{Ext}(\mathcal{AF}_\mathcal{K})$ **do**
    $S = \{a | a \in \mathcal{E}$ *such that* $\texttt{Conc}(a) \models Q\}$
  for all $\mathcal{E} \in \texttt{Ext}(\mathcal{AF}_\mathcal{K})$ *such that* $\nexists a \in \mathcal{E}$ *and* $\texttt{Conc}(a) \models Q$ **do**
    for all $a \in S$ **do**
      $Attakers = \{b | b \in \mathcal{E}$ *such that* b *attacks* $a\}$
      $\mathcal{C} = \mathcal{C} \cup \{Attakers\}$
  $\mathcal{B} = \bigcup_{S \in \mathcal{C}} S$
  Return $(\mathcal{C}, \mathcal{B})$

transforming any instance of the strong proponent set (resp. opponent set) problem to an instance of the hitting set problem.

Theorem 2. *[Reduction] Let $\mathcal{K}$ be a knowledge base, $\mathcal{AF}_\mathcal{K}$ the corresponding argumentation framework, $\texttt{Ext}(\mathcal{AF}_\mathcal{K}) = \{\mathcal{E}_1, ..., \mathcal{E}_m\}$ set of all preferred extensions of $\mathcal{AF}_\mathcal{K}$ and Q be a query.*

- *Suppose that $\mathcal{C} = \{S_1, ..., S_m\}$ and $\mathcal{B}$ are the outputs of the function **ReduceStrongToHitting** ($\texttt{Ext}(\mathcal{AF}_\mathcal{K}), Q$), then the set of arguments $\mathcal{H}$ is a strong proponent set of Q if and only if $\mathcal{H}$ is a minimal hitting set of $\mathcal{C}$.*
- *Suppose that $\mathcal{C} = \{S_1, ..., S_m\}$ and $\mathcal{B}$ are the outputs of the function **ReduceOpponentToHitting** ($\texttt{Ext}(\mathcal{AF}_\mathcal{K}), Q$), then the set of arguments $\mathcal{H}$ is an opponent set of Q iff $\mathcal{H}$ is a minimal hitting set of $\mathcal{C}$.*

Proof. In [1] pages 12-13.

Complexity of Algorithm 2 & 3. It is not hard to see that Algorithm 2 & Algorithm 3 run in a polynomial time $O(n^2)$ considering the worst-case scenario, in which $n = max(|\texttt{Ext}(\mathcal{AF}_\mathcal{K})|, |S|)$ where $|S|$ is the cardinality of the set of all arguments that support Q.

4.3 Computing Explanations

In order to compute an explanation we need to compute *one* strong proponent set (or opponent set) of Q. As a result of Theorem 2, computing *one* strong proponent set is at

Algorithm 4 COMPUTEMINHITTINGSET

1: **function** COMPUTEMINHITTINGSET($\mathcal{C}, \mathcal{B} = \{m_1, ..., m_n\}$)
2: $M = \mathcal{B}$
3: **for** $i \leftarrow 1, n$ **do**
4: $\mathcal{H} = M$-$\{m_j\}$
5: **if** $\forall S \in \mathcal{C}, \mathcal{H} \cap S \neq \emptyset$ **then** $M = \mathcal{H}$
6: **Return** (M)

Algorithm 5 COMPUTEEXPLANATION(Q,TYPE,$\texttt{Ext}(\mathcal{AF}_\mathcal{K})$)

Input: *The query to be explained, boolean variable* type *indicating the explanation type, the set of all extensions of the corresponding argumentation framework.*
Output: *An explanation.*

1: **if** $type = true$ **then** $(\mathcal{C}, \mathcal{B}) = ReduceStrongToHitting(\texttt{Ext}(\mathcal{AF}_\mathcal{K}), Q)$
2: **else**$(\mathcal{C}, \mathcal{B}) = ReduceOpponentToHitting(\texttt{Ext}(\mathcal{AF}_\mathcal{K}), Q)$
3: $\mathcal{H} = CompMinHittingSet(\mathcal{C}, \mathcal{B})$
4: $\mathcal{DT}(\mathcal{H}) = \{$*DefenseTree*$(a, \texttt{Ext}(\mathcal{AF}_\mathcal{K}))|a \in \mathcal{H}\}$
5: **Return** $(\mathcal{H}, \mathcal{DT}(\mathcal{H}))$

least as hard as computing *one* minimal hitting set. Based on the polynomial algorithm (Algorithm 4) for computing *one* minimal hitting set proposed in [13], we will compute a strong proponent set (or an opponent set) by means of the reduction algorithms (Algorithm 2 & 3).

Algorithm 5 computes an explanation for query failure and query acceptance. The boolean constant *true* corresponds to acceptance of Q and *false* corresponds to the failure of Q. The algorithm uses the reduction (lines 1 and 2) to make use of the hitting set algorithm (line 3). Finally, in line 4 the algorithm returns the explanation after having computed the defense trees of the set S. The complexity of *ComputeExplanation* depends on the complexity of *ReduceStrongToHitting()* (resp. *ReduceStrongToHitting()*), *CompMinHittingSet()* and *DefenseTree()* which is polynomial.

Computing all explanations is hard to perform since one has to find all strong proponent (resp. opponent) sets to be able to compute all explanations, and finding all strong proponent (resp. opponent) sets is at least as hard as find all minimal hitting sets.

Theorem 3. *Let $\mathcal{K}$ be an inconsistent knowledge base, Q a query and $\texttt{exp}^+(Q)$ (resp. $\texttt{exp}^-(Q)$) the set of all explanations of the acceptance (resp. failure) of Q. Then, computing $\texttt{exp}^+(Q)$ (resp. $\texttt{exp}^-(Q)$) is NP-hard.*

Proof. In [1] page 14.

Despite the hardness of computing all explanations, a brute-force algorithm can be constructed as follows. We use the reduction function *ReduceStrongToHitting()* and *ReduceOpponentToHitting()*. Next, we use Reiter's well-known hitting set algorithm [23, 15] as a subroutine called *ComputeAllMinHit()*. After that, given the reduction between the problem of hitting set and strong proponent (or opponent) set, this subroutine can

compute all strong proponent (or opponent) sets. Finally, for each strong proponent (or opponent) set we compute the set of defense trees of its arguments.

5 Discussion and conclusion

In recent years explanation has drawn a tremendous attention in the field of Description Logics, OWL Ontologies debugging and Database Systems. In the field of databases there has been work on explaining query answering and query failure [21] using causality and responsibility or cooperative architectures. In the area of DLs, the question was mainly about explaining either reasoning (subsumption and non-subsumption) or unsatisfiability and incoherence. In [19] the authors addressed the problem of explaining subsumption and non-subsumption in a coherent and satisfiable DL knowledge base using *formal proofs as explanation* while other proposals [24] have used *Axiom pinpointing* and *Concept pinpointing* as explanation to highlight contradictions within an unsatisfiable and incoherent DL KB. Another proposal [16] is the so-called *justification-oriented proofs* in which the authors proposed a *proof-like* explanation without the need for a deduction system.

In previous work [2, 3] we have proposed explanation facilities for inconsistent-knowledge base, we focused in these work on the representational aspects of explanation where a dialogue between the user and the system takes place to explain query failure, in this work we are more interested in the definition of the core explanation and the computational aspects, in addition we handle query answering in its broader sense, i.e. query failure and query acceptance.

Acknowledgement Financial support from the French National Research Agency (ANR) for the project DUR-DUR (ANR-13-ALID-0002) is gratefully acknowledged.

References

1. A. Arioua, N. Tamani, and M. Croitoru. Query answering explanation in inconsistent datalog+/- knowledge bases. Technical report, INRIA GraphiK - LIRMM, INRA, UM, 2014. http://www2.lirmm.fr/∼arioua/Arioua2014TR.pdf, please mind the tild.
2. A. Arioua, N. Tamani, and M. Croitoru. Query failure explanation in inconsistent knowledge bases an argumentation approach: Extended abstract. In *5th International Conference on Computational Models of Argument 2014, to appear*, 2014.
3. A. Arioua, N. Tamani, M. Croitoru, and P. Buche. Query failure explanation in inconsistent knowledge bases: A dialogical approach. In *Research and Development in Intelligent Systems XXXI*, pages 119–133. Springer, 2014.
4. F. Baader, S. Brandt, and C. Lutz. Pushing the el envelope. In *Proc. of IJCAI 2005*, 2005.
5. J.-F. Baget, M.-L. Mugnier, S. Rudolph, and M. Thomazo. Walking the complexity lines for generalized guarded existential rules. In *Proc. of IJCAI'11*, pages 712–717, 2011.
6. M. Bienvenu. On the complexity of consistent query answering in the presence of simple ontologies. In *Proc of AAAI*, 2012.
7. M. Bienvenu and R. Rosati. Tractable approximations of consistent query answering for robust ontology-based data access. In *Proc of IJCAI'13*, pages 775–781. AAAI Press, 2013.
8. A. Calì, G. Gottlob, and T. Lukasiewicz. A general datalog-based framework for tractable query answering over ontologies. *Web Semantics: Science, Services and Agents on the World Wide Web*, 14:57–83, 2012.

9. D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, and R. Rosati. Tractable reasoning and efficient query answering in description logics: The dl-lite family. *J. Autom. Reasoning*, 39(3):385–429, 2007.
10. D. Calvanese, M. Ortiz, M. Šimkus, and G. Stefanoni. Reasoning about explanations for negative query answers in dl-lite. *J. of Artificial Intelligence Research*, 48:635–669, 2013.
11. M. Croitoru and S. Vesic. What can argumentation do for inconsistent ontology query answering? In *Scalable Uncertainty Management*, volume 8078 of *Lecture Notes in Computer Science*, pages 15–29. Springer Berlin Heidelberg, 2013.
12. P. M. Dung. On the acceptability of arguments and its fundamental role in nonmonotonic reasoning, logic programming and n-person games. *Artificial intelligence*, 77(2):321–357, 1995.
13. A. Fijany and F. Vatan. New approaches for efficient solution of hitting set problem. In *Proceedings of the Winter International Synposium on Information and Communication Technologies*, WISICT '04. Trinity College Dublin, 2004.
14. G. Gottlob, A. Pieris, et al. Towards more expressive ontology languages: The query answering problem. *Artificial Intelligence*, 193:87–128, 2012.
15. R. Greiner, B. A. Smith, and R. W. Wilkerson. A correction to the algorithm in reiter's theory of diagnosis. *Artificial Intelligence*, 41(1):79–88, 1989.
16. M. Horridge, B. Parsia, and U. Sattler. Justification oriented proofs in owl. In *Proc. of ISWC 2010.*, pages 354–369. Springer-Verlag, 2010.
17. D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo. Inconsistency-tolerant semantics for description logics. In *Proceedings of the Fourth International Conference on Web Reasoning and Rule Systems*, RR'10, pages 103–117. Springer-Verlag, 2010.
18. T. Lukasiewicz, M. V. Martinez, and G. I. Simari. Complexity of inconsistency-tolerant query answering in datalog+/-. In *Proceedings of ODBASE 2013*, volume 8185 of *Lecture Notes in Computer Science*, pages 488–500. Springer, 2013.
19. D. L. McGuinness and A. T. Borgida. Explaining subsumption in description logics. In *Proceedings of IJCAI'95*, pages 816–821. Morgan Kaufmann Publishers Inc., 1995.
20. D. L. McGuinness and P. F. Patel-Schneider. Usability issues in knowledge representation systems. In *Proc. of AAAI-98*, pages 608–614, 1998.
21. A. Meliou, W. Gatterbauer, K. F. Moore, and D. Suciu. Why so ? or why no ? functional causality for explaining query answers. In *Proc. of the International Workshop on Management of Uncertain Data*, 2010.
22. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. In *Journal on data semantics X*, pages 133–173. Springer, 2008.
23. R. Reiter. A theory of diagnosis from first principles. *Artificial intelligence*, 32(1):57–95, 1987.
24. S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of IJCAI'03*, pages 355–360. Morgan Kaufmann Publishers Inc., 2003.
25. L. R. Ye and P. E. Johnson. The impact of explanation facilities on user acceptance of expert systems advice. *Mis Quarterly*, 19(2), 1995.