Evaluation of techniques for inconsistency handling in OWL 2 QL ontologies

Riccardo Rosati, Marco Ruzzi, Mirko Graziosi, Giulia Masotti

DIAG, Sapienza Università di Roma
Via Ariosto 25, I-00185 Roma, Italy

Abstract. In this paper we present the Quonto Inconsistent Data handler (QuID). QuID is a reasoner for OWL 2 QL that is based on the system Quonto and is able to deal with inconsistent ontologies. The central aspect of QuID is that it implements two different, orthogonal strategies for dealing with inconsistency: ABox repairing techniques, based on data manipulation, and consistent query answering techniques, based on query rewriting. Moreover, by exploiting the ability of Quonto to delegate the management of the ABox to a relational database system (DBMS), such techniques are potentially able to handle very large inconsistent ABoxes. For the above reasons, QuID allows for experimentally comparing the above two different strategies for inconsistency handling in the context of OWL 2 QL. We thus report on the experimental evaluation that we have conducted using QuID. Our results clearly point out that inconsistency-tolerance in OWL 2 QL ontologies is feasible in practical cases. Moreover, our evaluation singles out the different sources of complexity for the data manipulation technique and the query rewriting technique, and allows for identifying the conditions under which one method is more efficient than the other.

1 Introduction

One of the most important current issues in OWL ontology management is dealing with inconsistency, that is, the presence of contradictory information in the ontology [8]. It is well-known that the classical semantics of OWL and Description Logics (DL) is not *inconsistency-tolerant*, i.e., it does not allow for using in a meaningful way any piece of information in an inconsistent ontology. On the other hand, the size of ontologies used by real applications is scaling up, and ontologies are increasingly merged and integrated into larger ontologies: the probability of creating inconsistent ontologies is consequently getting higher and higher (see e.g. [4]).

In this paper we focus on *ABox inconsistency*, i.e., the case of inconsistent ontologies where the TBox (intensional part of the ontology) is consistent, while the ABox (extensional part of the ontology) is inconsistent with the TBox, i.e., a subset of the assertions in the ABox contradicts one or more TBox assertions.

We follow an approach that is formally based on inconsistency-tolerant semantics; such semantics overcome the limitations of the classical DL semantics in inconsistency management. In particular, we consider inconsistency-tolerant semantics for general DLs recently proposed in [5], called IAR *semantics*, for which reasoning has been studied in the context of the Description Logics of the *DL-Lite* family, and in particular the DL *DL-Lite*$_A$, that underlies the OWL profile OWL 2 QL. The IAR semantics is centered around the notion of *ABox repair*, which is a very simple and natural one: the ABox repair of a DL ontology is the intersection of all the maximal subsets of the ABox that are consistent with the TBox.

Recently, two different methods for reasoning under the IAR inconsistency-tolerant semantics have been studied: techniques based on the computation of the ABox repair (*ABox cleaning*) and techniques based on a tranformation of the queries posed to the (possibly inconsistent) ontology (*consistent query rewriting*). In particular, in [5] it was proved that computing the ABox repair of a *DL-Lite*$_A$ ontology under the IAR semantics is a tractable problem. Then, in [6] a technique for query answering under IAR-semantics in *DL-Lite*$_A$ is presented: instead of modifying the ABox, this method is based on computing a rewriting Q' of the initial query Q and then evaluating the query Q' with respect to the original ABox.

We argue that the results of [5, 6] are potentially very important from the practical viewpoint, for the following reasons: (i) they are based on formally grounded notions of inconsistency-tolerant semantics; (ii) they identify (to the best of our knowledge) the first inconsistency-tolerant semantics in DLs for which query answering is tractable. So, based on such results, in principle it might be possible to define practical algorithms for handling inconsistency in OWL 2 QL.

This paper starts from the above results, and tries to provide an experimental evaluation and comparison of both the ABox cleaning approach and the consistent query rewriting approach mentioned above. In particular, our main goal was to address the following fundamental questions: (i) is ABox cleaning a feasible technique? (ii) is consistent query rewriting a feasible technique? (iii) under which conditions consistent query rewriting is to prefer to ABox cleaning (and vice versa)?

In this paper, we provide the following contributions:

(1) We present effective techniques for both ABox cleaning and consistent query rewriting in *DL-Lite*$_A$/OWL 2 QL under IAR semantics. To this aim, we present the QUonto Inconsistent Data handler (QuID), that implements, within the Quonto system,[1] techniques for both the computation of the ABox repair of a *DL-Lite*$_A$ ontology under the above semantics, as well as techniques for computing the consistent query rewriting of queries. QuID constitutes (to the best of our knowledge) the first implementation of tractable algorithms for handling inconsistent instances in OWL ontologies. Moreover, Quonto delegates the management of the ABox to a relational database system (DBMS). Therefore, for ABox cleaning, all modifications of the ABox are delegated to the DBMS through SQL queries and updates; and for consistent query rewriting, the rewritten query can be directly executed by the DBMS on the original database. This potentially allows for handling inconsistency in very large ABoxes under both techniques.

(2) We present the results of a set of experiments that we have conducted using QuID. These results clearly show that ABox cleaning in *DL-Lite*$_A$ is actually scalable: QuID is able to efficiently compute the IAR repair of both complex and large ontologies, whose ABoxes contain up to millions of assertions and have hundreds of thousands of assertions inconsistent with the TBox. On the other hand, the results for the query answering

[1] `http://www.dis.uniroma.it/~quonto`

technique based on consistent query rewriting are in general less encouraging, since the structural complexity of the reformulated queries makes the whole query answering process slower than the approach based on ABox cleaning, although consistent query rewriting does not require pre-processing of the ABox.

(3) Our experimental results allow us to understand the actual impact of the different aspects involved in the computation of the ABox repair and in consistent query rewriting, and the limits and possibilities of the two approaches implemented in QuID.

The rest of the paper is organized as follows. In Section 2, we present a detailed algorithm for computing IAR repairs in *DL-Lite*$_A$. In Section 3, we briefly recall the algorithm presented in [6] for consistent query rewriting under IAR semantics in *DL-Lite*$_A$. In Section 4 we present the QuID system and report on the experimental evaluation we have conducted with QuID. Finally, in Section 5 we conclude the paper.

2 ABox cleaning technique for OWL 2 QL

We start by briefly recalling the DL *DL-Lite*$_A$ and the IAR semantics.

In this paper we consider DL ontologies specified in *DL-Lite*$_A$, a member of the *DL-Lite* family of tractable Description Logics [2, 1], which is at the basis of OWL 2 QL, one of the profiles of OWL 2, the ontology specification language of the World Wide Web Consortium (W3C). *DL-Lite*$_A$ distinguishes concepts from *value-domains*, which denote sets of (data) values, and roles from *attributes*, which denote binary relations between objects and values. Concepts, roles, attributes, and value-domains in this DL are formed according to the following syntax:

$$\begin{array}{ll} B \longrightarrow A \mid \exists Q \mid \delta(U) & E \longrightarrow \rho(U) \\ C \longrightarrow B \mid \neg B & F \longrightarrow \top_D \mid T_1 \mid \cdots \mid T_n \\ Q \longrightarrow P \mid P^- & V \longrightarrow U \mid \neg U \\ R \longrightarrow Q \mid \neg Q & \end{array}$$

In such rules, A, P, and U respectively denote an atomic concept (i.e., a concept name), an atomic role (i.e., a role name), and an attribute name, P^- denotes the inverse of an atomic role, whereas B and Q are called basic concept and basic role, respectively. Furthermore, $\delta(U)$ denotes the *domain* of U, i.e., the set of objects that U relates to values; $\rho(U)$ denotes the *range* of U, i.e., the set of values that U relates to objects; $\top_D$ is the universal value-domain; $T_1, \ldots, T_n$ are n pairwise disjoint unbounded value-domains. A *DL-Lite*$_A$ ontology is a pair $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$, where $\mathcal{T}$ is the TBox and $\mathcal{A}$ the ABox. The TBox $\mathcal{T}$ is a finite set of assertions of the form

$$B \sqsubseteq C \qquad Q \sqsubseteq R \qquad E \sqsubseteq F \qquad U \sqsubseteq V \qquad (\mathsf{funct}\ Q) \qquad (\mathsf{funct}\ U)$$

From left to right, the first four assertions respectively denote inclusions between concepts, roles, value-domains, and attributes. In turn, the last two assertions denote functionality on roles and on attributes. In fact, in *DL-Lite*$_A$ TBoxes we further impose that roles and attributes occurring in functionality assertions cannot be specialized (i.e., they cannot occur in the right-hand side of inclusions). In practice, the only difference between *DL-Lite*$_A$ and OWL 2 QL lies in the presence of functionality assertions (which are not allowed in OWL 2 QL). Due to space limitations, we refer the reader to [7] for details on the semantics of *DL-Lite*$_A$.

We then briefly recall the IAR semantics for inconsistency-tolerance in DL ontologies (see [5] for more details). Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL ontology. Then, the IAR*-repair* of $\mathcal{K}$ is defined as the ABox corresponding to the intersection of all the maximal subsets of $\mathcal{A}$ that are consistent with $\mathcal{T}$. A first-order formula ϕ is entailed by $\mathcal{K}$ under the IAR semantics if ϕ is entailed by $\langle \mathcal{T}, \mathcal{A}_R \rangle$ under the standard DL semantics, where $\mathcal{A}_R$ is the IAR-repair of $\mathcal{K}$. We are interested in checking (Boolean) unions of conjunctive queries (UCQs) over DL ontologies.

The technique for computing the IAR-repair of a *DL-Lite*$_A$ ontology $\langle \mathcal{T}, \mathcal{A} \rangle$ is based on the idea of deleting from $\mathcal{A}$ all the membership assertions participating in *minimal* conflict sets for $\mathcal{T}$. As shown in [5], this task is relatively easy (in particular, tractable) in *DL-Lite*$_A$ because the following property holds: for every *DL-Lite*$_A$ TBox $\mathcal{T}$, all the minimal conflict sets for $\mathcal{T}$ are either unary conflict sets or binary conflict sets. This property is actually crucial for tractability of reasoning under IAR semantics.

We now present a detailed algorithm for computing the IAR-repair of a *DL-Lite*$_A$ ontology. This algorithm exploits the techniques presented in [5], whose aim was only to provide PTIME upper bounds for the problem of computing such repairs. In particular, the present algorithms specify efficient ways of detecting minimal conflict sets. Instead, the previous techniques check all unary and binary subsets of the ABox for these purposes.

In the following, we call *annotated ABox assertion* an expression ξ of the form $\langle \alpha, \gamma \rangle$ where α is an ABox assertion and γ is a value in the set $\{cons, ucs, bcs\}$. Furthermore, we call *annotated ABox* a set of annotated ABox assertions. The intuition behind an annotated ABox assertion ξ is that its annotation γ expresses whether the associated ABox expression α does not participate in any minimal conflict set (*cons*) or participates in a unary conflict set (*ucs*) or to a binary conflict set (*bcs*).

The following algorithm QuID-IAR-repair computes the IAR-repair of a *DL-Lite*$_A$ ontology. For ease of exposition, the algorithm does not report details on the treatment of attributes, which are actually handled in a way analogous to roles. In the following, we denote concept names with the symbol A, role names with the symbol P, basic concepts (that is, a concept name A or the domain of a role $\exists P$ or the range of a role $\exists P^-$) with the symbols B_1, B_2, and basic roles (that is, either a role name P or the inverse of a role name P^-) with the symbols R, S. Moreover, the expression $B(a)$ with B basic concept denotes: the instance assertion $A(a)$ if $B = A$; an instance assertion of the form $P(a, b)$ if $B = \exists P$; an instance assertion of the form $P(b, a)$ if $B = \exists P^-$.

```
**Algorithm** QuID-IAR-repair($\mathcal{K}$)
**input**: *DL-Lite*$_A$ ontology $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$,   **output**: $IAR$-repair of $\mathcal{K}$
**begin**
// STEP 1: create annotated ABox $\mathcal{A}_{ann}$
  $\mathcal{A}_{ann} = \emptyset$;
  **for each** $\alpha \in \mathcal{A}$ **do** $\mathcal{A}_{ann} = \mathcal{A}_{ann} \cup \langle \alpha, cons \rangle$;
// STEP 2: detect unary conflict sets in $\mathcal{A}_{ann}$
  **for each** concept name $A$ s.t. $\mathcal{T} \models A \sqsubseteq \neg A$ **do**
    **for each** $\xi = \langle A(a), cons \rangle \in \mathcal{A}_{ann}$ **do** $\mathcal{A}_{ann} = \mathcal{A}_{ann} - \{\xi\} \cup \{\langle A(a), ucs \rangle\}$;
  **for each** role name $P$ s.t. $\mathcal{T} \models P \sqsubseteq \neg P$ **do**
```

```
        for each $\xi = \langle P(a,b), \mathit{cons}\rangle \in \mathcal{A}_{ann}$ do $\mathcal{A}_{ann} = \mathcal{A}_{ann} - \{\xi\} \cup \{\langle P(a,b), \mathit{ucs}\rangle\}$;
    for each role name $P$ s.t. $\mathcal{T} \models P \sqsubseteq \neg P^-$ or $\mathcal{T} \models \exists P \sqsubseteq \neg\exists P^-$ do
        for each $\xi = \langle P(a,a), \mathit{cons}\rangle \in \mathcal{A}_{ann}$ do $\mathcal{A}_{ann} = \mathcal{A}_{ann} - \{\xi\} \cup \{\langle P(a,a), \mathit{ucs}\rangle\}$;
// STEP 3: detect binary conflict sets in $\mathcal{A}_{ann}$
    for each disjointness $B_1 \sqsubseteq \neg B_2$ such that $\mathcal{T} \models B_1 \sqsubseteq \neg B_2$ do
        for each pair $\xi_1 = \langle B_1(a), \gamma_1\rangle, \xi_2 = \langle B_2(a), \gamma_2\rangle \in \mathcal{A}'_{ann}$ such that $\gamma_1, \gamma_2 \neq \mathit{ucs}$ do
            $\mathcal{A}_{ann} = \mathcal{A}_{ann} - \{\xi_1, \xi_2\} \cup \{\langle B_1(a), \mathit{bcs}\rangle, \langle B_2(a), \mathit{bcs}\rangle\}$;
    for each disjointness $R \sqsubseteq \neg S$ such that $\mathcal{T} \models R \sqsubseteq \neg S$ do
        for each pair $\xi_1 = \langle R(a,b), \gamma_1\rangle, \xi_2 = \langle S(a,b), \gamma_2\rangle \in \mathcal{A}'_{ann}$ such that $\gamma_1, \gamma_2 \neq \mathit{ucs}$ do
            $\mathcal{A}_{ann} = \mathcal{A}_{ann} - \{\xi_1, \xi_2\} \cup \{\langle R(a,b), \mathit{bcs}\rangle, \langle S(a,b), \mathit{bcs}\rangle\}$;
    for each functionality assertion (funct $R$) $\in \mathcal{T}$ do
        for each pair $\xi_1 = \langle R(a,b), \gamma_1\rangle, \xi_2 = \langle R(a,c), \gamma_2\rangle \in \mathcal{A}'_{ann}$
            such that $b \neq c$ and $\gamma_1, \gamma_2 \neq \mathit{ucs}$ do
            $\mathcal{A}_{ann} = \mathcal{A}_{ann} - \{\xi_1, \xi_2\} \cup \{\langle R(a,b), \mathit{bcs}\rangle, \langle R(a,c), \mathit{bcs}\rangle\}$;
// STEP 4: extract the IAR repair from $\mathcal{A}_{ann}$
    $\mathcal{A}' = \emptyset$;
    for each $\langle \alpha, \mathit{cons}\rangle \in \mathcal{A}_{ann}$ do $\mathcal{A}' = \mathcal{A}' \cup \{\alpha\}$;
    return $\mathcal{A}'$
end
```

The algorithm QuID-IAR-repair consists of four steps which can be informally described as follows.

step 1 *copy of $\mathcal{A}$ into an annotated ABox $\mathcal{A}_{ann}$*. In this step, the value of the annotation is initialized to *cons* for all ABox assertions.

step 2 *detection of the unary conflict sets in $\mathcal{A}_{ann}$*. For every assertion of the form $\xi = \langle \alpha, \mathit{cons}\rangle$, such that $\{\alpha\}$ is a unary conflict set for $\mathcal{T}$, $\mathcal{A}_{ann} = \mathcal{A}_{ann} - \{\xi\} \cup \{\langle \alpha, \mathit{ucs}\rangle\}$, i.e., the annotation relative to α is changed to *ucs*. Unary conflict sets are actually detected through TBox reasoning, by looking at empty concepts and roles in $\mathcal{T}$, as well as asymmetric roles, i.e., roles disjoint with their inverse.

step 3 *detection of the binary conflict sets in $\mathcal{A}_{ann}$*. For every pair of assertions of the form $\xi_1 = \langle \alpha_1, \gamma_1\rangle$, $\xi_2 = \langle \alpha_2, \gamma_2\rangle$ such that $\gamma_1 \neq \mathit{ucs}$ and $\gamma_2 \neq \mathit{ucs}$ and $\{\alpha, \beta\}$ is a binary conflict set for $\mathcal{T}$, $\mathcal{A}_{ann} = \mathcal{A}_{ann} - \{\xi_1, \xi_2\} \cup \{\langle \alpha, \mathit{bcs}\rangle, \langle \beta, \mathit{bcs}\rangle\}$, i.e., the annotation relative to α and β is changed to *bcs*. As in the case of unary conflict sets, to find binary conflict sets the algorithm looks for disjoint concepts and roles in $\mathcal{T}$, as well as functional roles.

step 4 *extraction of the IAR-repair from $\mathcal{A}_{ann}$*. The *IAR*-repair can be now simply extracted from the annotated ABox $\mathcal{A}_{ann}$, by eliminating both unary conflict sets and binary conflict sets. Therefore, for every assertion of the form $\langle \alpha, \mathit{cons}\rangle$ in $\mathcal{A}_{ann}$, α is copied into the (non-annotated) ABox $\mathcal{A}'$ which is finally returned by the algorithm.

Correctness of the above algorithm can be proved starting from the results in [5].

Theorem 1. *Let $\mathcal{K}$ be a DL-Lite$_A$ ontology and let $\mathcal{A}'$ be the ABox returned by QuID-IAR-repair($\mathcal{K}$). Then, $\mathcal{A}'$ is the IAR repair of $\mathcal{K}$.*

3 Perfect reformulation of UCQs under IAR semantics

We now briefly recall the query rewriting technique proposed in [6]. Such a technique computes a first-order query Q' starting from a union of conjunctive queries Q and a *DL-Lite*$_A$ TBox $\mathcal{T}$. The query Q' is a *perfect reformulation of* Q *with respect to* $\mathcal{T}$ *under the IAR semantics*, i.e., Q' is such that, for every ABox $\mathcal{A}$, the answers to Q over $\langle \mathcal{T}, \mathcal{A} \rangle$ under the IAR semantics correspond to to the answers to Q' computed over the ABox $\mathcal{A}$ only. Due to space limits, here we just report the main definitions of the query rewriting technique: we refer the reader to [6] for more details on the method.

The first definition that we give can be used to establish whether a certain atom is consistent with the TBox axioms. Let A be an atomic concept in $\Gamma_{\mathcal{O}}$ and t a term (i.e., either a constant or a variable symbol), we pose $\mathit{ConsAt}_A^{\mathcal{T}}(t) = \mathit{false}$ if $\mathcal{T} \models A \sqsubseteq \neg A$, true otherwise. That is, $\mathit{ConsAt}_A^{\mathcal{T}}(t)$ is false if and only if the concept A is unsatisfiable. For an atomic role $P \in \Gamma_{\mathcal{O}}$ and terms t, t', we define: (i) $\mathit{ConsAt}_P^{\mathcal{T}}(t, t') = \mathit{false}$ if $\mathcal{T} \models P \sqsubseteq \neg P$; (ii) $t \neq t'$ if $\mathcal{T} \models P \sqsubseteq \neg P^-$ or $\mathcal{T} \models \exists P \sqsubseteq \neg \exists P^-$; (iii) true otherwise (an analogous definition holds for an attribute $U \in \Gamma_{\mathcal{O}}$ and terms t and t').

Now we deal with possible clashes involving negative inclusions, which are also called *disjointnesses*. Let B be a basic concept built from an atomic concept or an atomic role of $\Gamma_{\mathcal{O}}$, and let t be a term. Then, we define $\mathit{NotDisjClash}_B^{\mathcal{T}}(t)$ as the following FOL formula:

$$\bigwedge_{A \in DCN(B,\mathcal{T})} \neg(A(t) \wedge \mathit{ConsAt}_A^{\mathcal{T}}(t)) \wedge \bigwedge_{P \in DRD(B,\mathcal{T})} \neg(\exists y.P(t,y) \wedge \mathit{ConsAt}_P^{\mathcal{T}}(t,y)) \wedge$$
$$\bigwedge_{P \in DRR(B,\mathcal{T})} \neg(\exists y.P(y,t) \wedge \mathit{ConsAt}_P^{\mathcal{T}}(y,t)) \wedge \bigwedge_{U \in DAD(B,\mathcal{T})} \neg(\exists y.U(t,y) \wedge \mathit{ConsAt}_U^{\mathcal{T}}(t,y))$$

where y is a variable symbol such that $y \neq t$, DCN, DRD, DRR, and DAD are defined as follows:

$$\begin{array}{l}
DCN(B,\mathcal{T}) = \{A \mid A \text{ is an atomic concept of } \Gamma_{\mathcal{O}} \text{ and } \mathcal{T} \models B \sqsubseteq \neg A\} \\
DRD(B,\mathcal{T}) = \{P \mid P \text{ is an atomic role of } \Gamma_{\mathcal{O}} \text{ and } \mathcal{T} \models B \sqsubseteq \neg \exists P\} \\
DRR(B,\mathcal{T}) = \{P \mid P \text{ is an atomic role of } \Gamma_{\mathcal{O}} \text{ and } \mathcal{T} \models B \sqsubseteq \neg \exists P^-\} \\
DAD(B,\mathcal{T}) = \{U \mid U \text{ is an attribute of } \Gamma_{\mathcal{O}} \text{ and } \mathcal{T} \models B \sqsubseteq \neg \delta(U)\}
\end{array}$$

Let us now consider disjointness clashes for roles. Let P be a role name from $\Gamma_{\mathcal{O}}$ and let t, t' be terms, we define the formula $\mathit{NotDisjClash}_P^{\mathcal{T}}(t, t')$ as follows:

$$\bigwedge\nolimits_{S \in \mathit{DisjRoles}(P,\mathcal{T})} \neg(S(t,t') \wedge \mathit{ConsAt}_S^{\mathcal{T}}(t,t')) \wedge \mathit{NotDisjClash}_{\exists P}^{\mathcal{T}}(t) \wedge$$

$$\bigwedge\nolimits_{S \in \mathit{DisjInvRoles}(P,\mathcal{T})} \neg(S(t',t) \wedge \mathit{ConsAt}_S^{\mathcal{T}}(t',t)) \wedge \mathit{NotDisjClash}_{\exists P^-}^{\mathcal{T}}(t')$$

where, again, if either t or t' are variable symbols, then they are free variables, and the sets $\mathit{DisjRoles}(P, \mathcal{T})$ and $\mathit{DisjInvRoles}(P, \mathcal{T})$ are defined as follows:

$$\begin{array}{l}
\mathit{DisjRoles}(P,\mathcal{T}) = \{S \mid S \text{ is a role name of } \Gamma_{\mathcal{O}} \text{ and } \mathcal{T} \models P \sqsubseteq \neg S\} \\
\mathit{DisjInvRoles}(P,\mathcal{T}) = \{S \mid S \text{ is a role name of } \Gamma_{\mathcal{O}} \text{ and } \mathcal{T} \models P \sqsubseteq \neg S^-\}.
\end{array}$$

Intuitively, $\mathit{NotDisjClash}_P^{\mathcal{T}}(t, t')$ will be used in the reformulation to deal with possible violations of negative inclusions involving P. This means considering role inclusions, through the sets $\mathit{DisjRoles}(P, \mathcal{T})$ and $\mathit{DisjInvRoles}(P, \mathcal{T})$, and concept inclusions of the form $\exists P \sqsubseteq \neg B$ and of the form $\exists P^- \sqsubseteq \neg B$, through the use of $NotDisjClash^{\mathcal{T}}_{\exists P}(t)$ and $NotDisjClash^{\mathcal{T}}_{\exists P^-}(t')$, respectively. $ConsAt^{\mathcal{T}}_{S}(t,t')$ plays here a role analogous to the one played by $ConsAt$ formulas in $NotDisjClash^{\mathcal{T}}_{B}(t)$. (The function $NotDisjClash^{\mathcal{T}}_{U}$ for attributes U is defined in an analogous way.)

Finally, we consider clashes on functionalities and define $NotFunctClash^{\mathcal{T}}_{P}(t,t')$ as the following FOL formula:

– if (funct P) $\notin \mathcal{T}$ and (funct P^-) $\notin \mathcal{T}$, then $NotFunctClash^{\mathcal{T}}_{P}(t,t') = true$;
– if (funct P) $\in \mathcal{T}$ and (funct P^-) $\notin \mathcal{T}$, then $NotFunctClash^{\mathcal{T}}_{P}(t,t') = \neg(\exists y.P(t,y) \wedge y \neq t' \wedge ConsAt^{\mathcal{T}}_{P}(t,y))$;
– if (funct P) $\notin \mathcal{T}$ and (funct P^-) $\in \mathcal{T}$, then $NotFunctClash^{\mathcal{T}}_{P}(t,t') = \neg(\exists y.P(y,t') \wedge y \neq t \wedge ConsAt^{\mathcal{T}}_{P}(y,t))$;
– if (funct P) $\in \mathcal{T}$ and (funct P^-) $\in \mathcal{T}$, then $NotFunctClash^{\mathcal{T}}_{P}(t,t') = \neg(\exists y.P(t,y) \wedge y \neq t' \wedge ConsAt^{\mathcal{T}}_{P}(t,y)) \wedge \neg(\exists y.P(y,t') \wedge y \neq t \wedge ConsAt^{\mathcal{T}}_{P}(y,t))$.

(The function $NotFunctClash^{\mathcal{T}}_{U}$ for attributes U is defined analogously.)

We are now able to define for each *DL-Lite*$_A$ construct the formula that combines together the various formulas we have introduced for dealing with the various possible clashes: (i) $NotClash^{\mathcal{T}}_{A}(t) = NotDisjClash^{\mathcal{T}}_{A}(t)$ for an atomic concept name A and term t; (ii) $NotClash^{\mathcal{T}}_{Z}(t,t') = NotDisjClash^{\mathcal{T}}_{Z}(t,t') \wedge NotFunctClash^{\mathcal{T}}_{Z}(t,t')$ for a role or attribute name Z and terms t,t'.

Let q be a CQ $\exists x_1,\ldots,x_k. \bigwedge_{i=1}^{n} A_i(t_i^1) \wedge \bigwedge_{i=1}^{m} P_i(t_i^2,t_i^3) \wedge \bigwedge_{i=1}^{\ell} U_i(t_i^4,t_i^5)$, where every A_i is an atomic concept, every P_i is an atomic role, every U_i is an attribute, and every $t_i^1,t_i^2,t_i^3,t_i^4,t_i^5$ is either a constant or a variable x_j with $1 \leq j \leq k$. Then, we define *IncRewriting*$_{IAR}(q,\mathcal{T})$ as the following FOL sentence

$$\begin{aligned}&\exists x_1,\ldots,x_k. \textstyle\bigwedge_{i=1}^{n} A_i(t_i^1) \wedge ConsAt^{\mathcal{T}}_{A_i}(t_i^1) \wedge NotClash^{\mathcal{T}}_{A_i}(t_i^1) \wedge\\ &\textstyle\bigwedge_{i=1}^{m} P_i(t_i^2,t_i^3) \wedge ConsAt^{\mathcal{T}}_{P_i}(t_i^2,t_i^3) \wedge NotClash^{\mathcal{T}}_{P_i}(t_i^2,t_i^3)\\ &\textstyle\bigwedge_{i=1}^{\ell} U_i(t_i^4,t_i^5) \wedge ConsAt^{\mathcal{T}}_{U_i}(t_i^4,t_i^5) \wedge NotClash^{\mathcal{T}}_{U_i}(t_i^4,t_i^5)\end{aligned}$$

Informally, for each atom $A_i(t_i^1)$, each membership assertion of the ABox $\mathcal{A}$ constituting an image of $A_i(t_i^1)$ has not to be inconsistent with the TBox (condition $ConsAt^{\mathcal{T}}_{A_i}(t_i^1)$), and has not to be involved in any clash with some other assertion of $\mathcal{A}$ on any negative inclusion (condition $NotClash^{\mathcal{T}}_{A_i}(t_i^1)$). Similarly for atoms of the form $P_i(t_i^2,t_i^3)$ and $U_i(t_i^4,t_i^5)$.

Let Q be the UCQ $q_1 \vee \ldots \vee q_n$. Then, we define *IncRewritingUCQ*$_{IAR}(Q,\mathcal{T}) = \bigvee_{i=1}^{n}$ *IncRewriting*$_{IAR}(q_i,\mathcal{T})$. Finally, we define *PerfectRef*$_{IAR}(Q,\mathcal{T})$ as *IncRewritingUCQ*$_{IAR}$(*PerfectRef*$(Q,\mathcal{T}),\mathcal{T})$, where *PerfectRef*$(Q,\mathcal{T})$ denotes the algorithm for computing a perfect reformulation of a UCQ Q with respect to a *DL-Lite*$_A$ TBox $\mathcal{T}$ under standard semantics [2,7] (the algorithm *PerfectRef*$(Q,\mathcal{T})$ returns a UCQ specified over $\mathcal{T}$). It can be shown (see [6]) that *PerfectRef*$_{IAR}(Q,\mathcal{T})$ constitutes a perfect reformulation of Q with respect to $\mathcal{T}$ under IAR semantics.

Therefore, using this technique, it is possible to solve query answering under IAR semantics in *DL-Lite*$_A$ as follows. Given the initial query Q and the ontology $\langle\mathcal{T},\mathcal{A}\rangle$, the first-order query *PerfectRef*$_{IAR}(Q,\mathcal{T})$ is computed, and then such a first-order query is evaluated over the original ABox (which is in general inconsistent with $\mathcal{T}$). So, in this case no repair of the ABox is performed, differently from the algorithm presented in the previous section.

4 Experiments

We have implemented the techniques presented in the previous Section in the Quonto system, in a module called QuID (the QUonto Inconsistent Data handler). Essentially, QuID is a Java implementation of the above algorithms for ABox repair and for query rewriting. In fact, in the Quonto architecture, the management of the ABox is delegated to a relational database management system (DBMS): therefore, all the operations on ABox assertions of the algorithms for computing repairs are executed in QuID by the DBMS used by Quonto, through appropriate SQL scripts.

We have experimented QuID in order to answer several open questions about: (i) the computational cost of the various steps of the ABox cleaning algorithm and of the query rewriting algorithm; (ii) the scalability of such algorithms; (iii) the impact of the "degree of inconsistency" of the ABox on the computational cost of the algorithms; (iv) the practical difference between the ABox cleaning tecnhique and the purely intensional rewriting technique.

Experimenting the QuID-IAR-repair algorithm We have experimented our implementation of the QuID-IAR-repair algorithm over the LUBM benchmark ontology,[2] whose TBox has 43 concept names, 25 role names, 7 attribute names, and about 200 TBox assertions. We have generated 4 different ABoxes by means of the UBA Data Generator provided by the LUBM website, with an increasing number of assertions, and used such ABoxes in our experiments. It is important to note that the original LUBM ontology has no axioms which can generate inconsistency, and hence, no inconsistent data is contained in the generated ABoxes. So, we sligthly modified the LUBM ontology by adding some "inconsistency-generating" axioms and then added inconsistencies to the ABoxes. We created four different version for every original ABox with different percentages of ABox assertions involved in minimal conflict sets, in order to get ABoxes with respectively $1\%, 5\%, 10\%$ and 20% of inconsistent assertions, uniformly distributed among the axioms which might generate inconsistency. Figure 1 shows the size (number of instance assertions) of the ABoxes we used in the experiments: every column is labeled with the number of Universities the ABox data contains, and every row is labeled with the percentage of inconsistent facts added to the ABox itself.

Figure 2 report some of the experimental results that we have obtained. The table displayed presents the experimental results for QuID-IAR-repair using a PostgreSQL 9.1 instance as external DBMS. The results have been conducted on a Pentium i5 (2.4 GHz) CPU with 4GB RAM under Windows 7 (64 bit) operating system.

All the necessary software, as well as instructions on how to reproduce the experiments presented in this section, are publicly available at `http://www.dis.uniroma1.it/~ruzzi/quid/`. Further details on the ontology used in the experiments are also available there.

In the table displayed in Figure 2, the first column reports the number of universities represented in the ABox, while the second column reports the percentage of ABox assertions that participate in minimal conflict sets for the considered TBox. Moreover:

- T1 denotes the time to create the annotated ABox (step 1 of QuID-IAR-repair);

[2] `http://swat.cse.lehigh.edu/projects/lubm/`

– T2 denotes the time to detect unary and binary conflict sets (steps 2 and 3 of QuID-IAR-repair);
– T3 denotes the time to extract the IAR-repair from the annotated ABox (step 4 of QuID-IAR-repair);
– Total is the total time to compute the IAR-repair, i.e., T1+T2+T3.

		Number of Universities			
		1	5	10	20
Inc. Perc.	1	103765	631960	1285244	2711216
	5	109165	658980	1339304	2819337
	10	115845	692400	1406124	2952957
	20	130445	765380	1552104	3244937

Fig. 1. Size of the UBA generated ABoxes

#Univ	Inc%	T1 (ms)	T2 (ms)	T3 (ms)	Total (ms)
1	1	66908	2356	73617	142881
	5	69748	11559	71401	152708
	10	71402	24523	70231	166156
	20	85878	50014	68156	204048
5	1	414477	13416	418970	846863
	5	419298	60434	414854	894586
	10	412371	131805	403619	947795
	20	466363	254000	406880	1127243
10	1	968123	31060	953037	1952220
	5	945471	140447	917890	2003808
	10	936688	271830	884835	2093353
	20	987216	573020	873664	2433900
20	1	2381829	137327	2379121	4898277
	5	2485267	353486	2251335	5090088
	10	2233066	722468	2212381	5167915
	20	2297791	1417200	2090794	5805785

Fig. 2. Repair generation time

The above experimental results show that:

(i) the computation of the IAR-repair (column T1) seems really scalable, and grows almost linearly w.r.t. the size of the ABox.
(ii) the percentage of inconsistency, i.e., the fraction of ABox assertions that participate in minimal conflict sets, has a real impact only on the detection of minimal conflict sets (column T2);
(iii) most of the whole execution time of the QuID-IAR-repair algorithm is devoted to the creation of annotated ABox (T1) and of the final repair (T3): if this could be avoided (e.g., by just modifying the original database, as explained below), the algorithm would be much more efficient, since only time T2 would be consumed.

Experimenting the consistent query rewriting approach As above observed, most of the execution time of the algorithm QuID-IAR-repair using a disk-resident DB is due to the creation of the annotated ABox (step 1) and to the creation of the IAR-repair (step 4). Thus, avoiding these steps would dramatically improve the efficiency of this algorithm.

To this aim, we observe that both the above steps could be completely avoided if the database schema used for representing the ABox would present an additional attribute for storing annotations in every relation (the usual DB representation of an ABox uses a unary relation for every concept and a binary relation for every role). This corresponds to the idea of directly using an annotated ABox instead of a standard ABox in the system. In this case, the computation of the IAR-repair could only consist of steps 2 and 3 of the algorithm QuID-IAR-repair. However, the choice of using an annotated ABox instead of a standard ABox could affect query answering, since the queries evaluated on an annotated ABox should be able to only consider the assertions whose annotation is equal to *cons*. Similarly, exploiting the query rewriting technique presented in Section 3, it is possible to completely avoid the computation of the annotated ABox, and could be able to evaluate the first-order query corresponding to the perfect reformulation of the original query directly over the original, inconsistent, ABox.

We have experimented whether this choice is actually feasible. In particular, we tested and compared three different approaches: (IAR) evaluation of the IAR perfect reformulation over the inconsistent ABox; (ANN) evaluation over the annotated ABox $\mathcal{A}_{ann}$ (produced by the QuID-IAR-repair algorithm) of the original query enriched with suitable conditions that are needed to filter out the assertions belonging to minimal conflict sets; (REP) evaluation of the original query over the repair using the standard query answering technique of QuOnto. Figure 3 presents a table showing the evaluation time of nine of the fourteen queries of the LUBM benchmark over all the ABoxes previously considered. We adopted a timeout (denoted by T.o. in the table) of 1 hour.[3]

Comparing the two approaches These experimental results show that, in QuID, evaluating queries on the annotated ABox is computationally not harder than evaluating them on the standard ABox. Conversely, the evaluation of the IAR perfect reformulations is often more expensive (in particular, it is more expensive for queries Q5–Q9). This is due to the fact that we have built no repair and we are querying the inconsisent ABox: thus, as shown in the previous section, the IAR perfect reformulation essentially has to select only assertions of the ABox which do not participate in minimal inconsistent sets (with respect to the TBox). This makes the form of such queries quite involved: in particular, the SQL queries corresponding to the IAR perfect refomulations of UCQs may present several nesting levels, which makes such queries hard to evaluate by current DBMSs. This consideration is enforced, e.g., by the evaluation time of query Q5, which is greater than 1 hour on the ABox representing 5 universities. That is, in this case the time to evaluate the IAR perfect reformulation of this query over the original ABox is much greater than computing the IAR repair and then evaluating the original query on the repaired ABox.

Combining the results of Figure 2 and Figure 3, it seems that, in general, the ABox cleaning approach is more convenient than the consistent query rewriting approach. In other words, the cost of preprocessing the ABox is generally an acceptable one,

[3] Further details on our experiments can be found at `http://www.dis.uniroma1.it/~ruzzi/quid/`.

		Q1			Q2			Q3		
#Univ	Inc%	IAR	ANN	REP	IAR	ANN	REP	IAR	ANN	REP
1	1	2324	37	31	31	2	0	32	7	16
	5	2340	36	16	32	3	0	31	8	0
	10	2325	40	16	31	0	0	31	10	0
	20	2340	36	16	32	3	0	31	7	0
5	1	905	2393	2808	31	238	405	63	874	999
	5	874	2882	3135	16	162	297	62	964	936
	10	561	3826	2668	32	113	218	78	882	936
	20	942	2968	4259	31	423	390	63	1063	1435
10	1	6661	11659	9531	32	162	390	3510	2441	2434
	5	4306	10878	9610	32	355	281	2122	2111	1966
	10	5663	8928	6926	47	437	406	1856	1977	2074
	20	4540	7677	7425	62	367	281	2871	1694	1701
20	1	11170	13625	20748	32	210	375	1639	1060	3229
	5	9859	18356	21887	63	317	390	2028	2198	3323
	10	9844	16870	14883	47	365	249	2075	2605	2996
	20	8783	18347	15725	63	482	296	2496	1486	2402

		Q4			Q5			Q6		
#Univ	Inc%	IAR	ANN	REP	IAR	ANN	REP	IAR	ANN	REP
1	1	78	4	0	4898	17	15	297	7	16
	5	78	0	15	4899	20	16	312	10	0
	10	78	0	0	4696	10	15	312	0	0
	20	62	4	15	4727	18	16	312	7	0
5	1	125	446	453	T.o.	15998	17659	1529	35	47
	5	203	351	421	T.o.	23721	14914	1529	37	47
	10	187	348	561	T.o.	15243	16521	1560	31	31
	20	202	444	749	T.o.	19612	22963	1748	30	15
10	1	3588	220	1372	T.o.	54142	52434	2995	66	62
	5	889	912	1185	T.o.	41709	54600	3167	66	62
	10	1701	237	936	T.o.	50099	48875	3229	70	63
	20	1607	771	843	T.o.	35762	40138	3448	73	78
20	1	1965	1398	1794	T.o.	99222	127796	6396	157	156
	5	2106	1121	1435	T.o.	112814	132288	6536	152	141
	10	2262	1238	1357	T.o.	113969	110622	6739	157	156
	20	3900	1273	1544	T.o.	103710	110339	7332	145	140

		Q7			Q8			Q9		
#Univ	Inc%	IAR	ANN	REP	IAR	ANN	REP	IAR	ANN	REP
1	1	4539	37	31	499	8	16	219	3	0
	5	4695	40	47	515	0	0	187	0	15
	10	4586	40	31	500	10	16	171	0	0
	20	4571	38	47	406	9	0	140	2	0
5	1	4652	557	453	1716	43	31	172	3	31
	5	4664	575	343	1731	39	31	156	29	0
	10	4648	533	515	1732	57	31	156	9	47
	20	4665	828	671	1731	40	16	141	30	31
10	1	4901	799	593	4025	71	62	234	17	16
	5	4790	894	655	3479	80	63	219	20	31
	10	4695	813	686	3339	88	63	234	18	15
	20	4477	833	483	3354	85	63	156	13	15
20	1	4508	590	577	5444	160	141	296	15	15
	5	4509	898	733	5553	160	140	312	10	16
	10	4430	753	437	5974	162	141	297	21	15
	20	4508	839	437	12215	171	156	156	9	16

Fig. 3. Query answering time (in milliseconds) for the various techniques

and really pays off during the evaluation of the queries, especially when the annotated representation of the ABox is adopted.

On the other hand, it is worth recalling that the ABox cleaning approach might not always be possible or easily realizable in real applications, especially in ontology-based data access (OBDA) scenarios where the ABox is actually a virtual object that is defined through virtual queries/views over one or more remote databases: (see e.g., [7]): in these cases, the OBDA system can typically only read such databases.

5 Conclusions

In this paper we have presented a practical approach to automatic the repair of inconsistent ontologies. The key features of our approach are the following: (i) the semantics of the repair are simple, intuitive, formally grounded, and defined for all DLs; (ii) such semantics allow for tractable automatic ABox cleaning and consistent query rewriting in the case of OWL 2 QL ontologies; (iii) our experiments show that the approach is really scalable, and that very large ABoxes can be effectively repaired.

The work presented in this paper can be extended in several directions. First, the present implementation can be certainly further optimized. For instance, besided working with an annotated ABox representation, other optimizations are possible: one possibility which seems worth exploring is employing summarization techniques for ABox representation, as in [3]. Also, the consistent query rewriting technique can be certainly optimized to the aim of reducing the size of the reformulated query. Then, it would be very interesting to see whether the techniques presented in this paper can be extended to other tractable OWL profiles.

Acknowledgments. This research has been partially supported by the ICT Collaborative Project ACSI (Artifact-Centric Service Interoperation), funded by the EU under FP7 ICT Call 5, 2009.1.2, grant agreement n. FP7-257593.

References

1. A. Artale, D. Calvanese, R. Kontchakov, and M. Zakharyaschev. The *DL-Lite* family and relations. *J. of Artificial Intelligence Research*, 36:1–69, 2009.
2. 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. of Automated Reasoning*, 39(3):385–429, 2007.
3. J. Dolby, J. Fan, A. Fokoue, A. Kalyanpur, A. Kershenbaum, L. Ma, J. W. Murdock, K. Srinivas, and C. A. Welty. Scalable cleanup of information extraction data using ontologies. In *ISWC/ASWC*, pages 100–113, 2007.
4. A. Hogan, A. Harth, A. Passant, S. Decker, and A. Polleres. Weaving the pedantic web. In *Proc. of 3rd Int. Workshop on Linked Data on the Web (LDOW2010)*, 2010.
5. D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo. Inconsistency-tolerant semantics for description logics. In *Proc. of RR 2010*, 2010.
6. D. Lembo, M. Lenzerini, R. Rosati, M. Ruzzi, and D. F. Savo. Query rewriting for inconsistent DL-Lite ontologies. In *Proc. of RR 2011*, 2011.
7. A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati. Linking data to ontologies. *J. on Data Semantics*, X:133–173, 2008.
8. Z. Wang, K. Wang, and R. W. Topor. A new approach to knowledge base revision in *DL-Lite*. In *Proc. of AAAI 2010*. AAAI Press, 2010.