Efficient Federated Debugging of Lightweight Ontologies

- Extended Version -

Andreas Nolle*, Christian Meilicke°, Heiner Stuckenschmidt°, and German Nemirovski*

* Business Informatics, Department of Business and Computer Science
Albstadt-Sigmaringen University, Germany
{nolle,nemirovskij}@hs-albsig.de
° Knowledge Representation and Knowledge Management Group,
Computer Science Institute, University of Mannheim, Germany
{chistian,heiner}@informatik.uni-mannheim.de

Abstract. In the last years ontologies have been applied increasingly as a conceptual view facilitating the federation of numerous data sources using different access methods and data schemes. Approaches such as ontology-based data integration (OBDI) are aimed at this purpose. According to these approaches, queries formulated in an ontology describing the knowledge domain as a whole are translated into queries formulated in vocabularies of integrated data sources. In such integrative environments the increasing number of heterogeneous data sources increases the risk of inconsistencies. These inconsistencies become a serious obstacle for leveraging the full potential of approaches like OBDI since inconsistencies can be hardly identified by existing reasoning algorithms, which mostly have been developed for processing of locally available knowledge bases. In this paper we present an alternative approach for efficient federated debugging. Our solution relies on the generation of so called clash queries that are evaluated over all integrated data sources. We further explain how these queries can be used for pinpointing those assertions that cause inconsistencies and discuss finally some experimental evaluation results of our implementation.

Keywords: Inconsistency Detection, Clash Queries, *DL-Lite*$_\mathcal{A}$, Federated Querying, Ontology-based Data Integration (OBDI), Query Rewriting, Backward-Chaining

1 Introduction

Dealing with distributed and heterogeneous data sources has become an important research topic since the amount of available data grows continuously in companies and in the public sector. To handle the resulting challenges of data

This extended version arises from our contribution to the 8th International Conference on Web Reasoning and Rule Systems (RR2014). The final publication is available at link.springer.com.

integration the approach of *ontology-based data access* (OBDA) has been proposed. In OBDA an ontology serves as conceptual view that comprises and possibly extends the semantics of each integrated data source. Mappings between this conceptual view and the different data schemes that describe the diverse data sources are used to transform original queries referring to the ontology into queries referring to the related vocabulary of each data source. Thus, on formulating queries clients do not have to be aware of each specific data schema. In the traditional OBDA approach, the data sources itself are assumed to be relational databases that are accessible via SQL. However, the approach of OBDA can also be adapted to all kinds of data sources and is in this setting also known under the designation *ontology-based data integration* (OBDI) [2, 18, 24].

Given in this context a set of distributed, heterogeneous *DL-Lite*$_\mathcal{A}$ knowledge bases. Even though each data source is self-consistent, the integrative knowledge base over all (or some of) these distributed sources may contain inconsistencies. We will illustrate this by the following example. Given a central ontology $\mathcal{T}$ and two distributed data sources DS_A and DS_B. For the sake of simplicity we assume that DS_A and DS_B use the same ontology $\mathcal{T}$. Note that our example can easily be extended to the case where DS_A and DS_B use different terminologies that are linked by equivalence or subsumption axioms in the central ontology $\mathcal{T}$.

Example 1. Our terminology $\mathcal{T}$ contains the following axioms that describe persons and their blood relationships:

$$
\begin{array}{rr}
Woman \sqsubseteq Person & \exists hasRelative \sqsubseteq Person \\
Man \sqsubseteq Person & \exists hasRelative^- \sqsubseteq Person \\
Woman \sqsubseteq \neg Man & hasAncestor \sqsubseteq hasRelative \\
(\mathsf{funct\ hasBirthday}) & hasDescendant \sqsubseteq hasRelative \\
\rho(\mathsf{hasBirthday}) \sqsubseteq \mathtt{xsd{:}dateTime} & hasDescendant \sqsubseteq hasAncestor^- \\
Person \sqsubseteq \delta(\mathsf{hasBirthday}) & hasAncestor \sqsubseteq \neg hasAncestor^- \\
(\mathsf{funct}\ hasDNA) & hasDescendant \sqsubseteq \neg hasDescendant^- \\
(\mathsf{funct}\ hasDNA^-) & gaveBirthTo \sqsubseteq hasDescendant \\
\exists hasDNA^- \sqsubseteq DNA & Person \sqsubseteq \exists hasAncestor \\
Person \sqsubseteq \exists hasDNA & \exists gaveBirthTo \sqsubseteq Woman
\end{array}
$$

The two data sources mentioned above contain the following assertions:

DS_A	DS_B
$Man(\mathbf{Homer})$	$gaveBirthTo(\mathbf{Homer}, \mathbf{Lisa})$
$Man(\mathbf{Bart})$	$gaveBirthTo(\mathbf{Marge}, \mathbf{Lisa})$
$Woman(\mathbf{Lisa})$	$hasRelative(\mathbf{Maggie}, \mathbf{Lisa})$
$Woman(\mathbf{Marge})$	...

Since in DS_B **Homer** is defined as someone who *gaveBirthTo* somebody, according to $\mathcal{T}$ **Homer** is implicitly defined to be a *Women*. However, at the same time we have $Man(\mathbf{Homer}) \in \mathrm{DS}_A$. Due to $Woman \sqsubseteq \neg Man \in \mathcal{T}$ we have obviously a contradiction between DS_A and DS_B.

To detect such contradictions the data of each integrated data source needs to be taken into account. Using traditional approaches, like tableau-based reasoning algorithms, requires to have all the data at a place before the algorithm can be applied. This makes such approaches hardly applicable in the context of huge amounts of distributed data. To facilitate identification of contradictions in distributed data sources, we propose an alternative approach. We identify all possible types of inconsistencies and formulate appropriate queries in terms of the central ontology. Furthermore, we reformulate these queries in order to take into account all logical consequences for each of the concepts, roles and attributes addressed in these queries and evaluate the rewritten queries, more precisely its query atoms at each integrated data source. To enable high efficiency of reasoning tasks and query answering we exploit a specific family of Description Logics, called *DL-Lite*, which has been especially developed for this aim.

The rest of the paper is organized as follows. In Section 2 we introduce some terms and definitions, especially for inconsistencies in description logics (Section 2.1), conjunctive queries over knowledge bases (Section 2.2), and *DL-Lite*$_\mathcal{A}$ and its family (Section 2.3), that are essential for our approach. In Section 3.1 we describe the task of inconsistency detection in *DL-Lite*$_\mathcal{A}$ knowledge bases. Sections 3.2 and 3.3 shows our approach for clash query generation and federation of clash queries, correspondingly. In Section 3.4 we further describe and prove an algorithm for inconsistency detection and generation of its explanations. In Section 4 we discuss some experimental evaluation results. Before concluding this paper in Section 6, we compare approaches related to our work in Section 5.

2 Preliminaries

In this section we, at first, explain some basic notions related to inconsistencies in knowledge bases. Then, we talk about conjunctive queries. Finally, we recall the characteristics of *DL-Lite*$_\mathcal{A}$.

2.1 Inconsistency in Description Logics

In Description Logics (DLs) [1] a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ consists of a TBox $\mathcal{T}$, the intensional knowledge part, and an ABox $\mathcal{A}$, the extensional knowledge part. Sets of objects can be denoted in terms of *concepts*; binary object relations correspond to *roles*; and binary relations between objects and values correspond to *attributes*. Expressions, assertions, and axioms in the knowledge base are built in a specific DL $\mathcal{L}$ over a signature (also known as alphabet or vocabulary) Σ, that comprises the set of all symbols for concepts, roles and attributes.

The semantics of a DL knowledge base $\mathcal{K}$ is defined in terms of interpretations and models. An interpretation is a pair of $\mathcal{I} = (\Delta^\mathcal{I}, \cdot^\mathcal{I})$ where $\Delta^\mathcal{I}$ represents the non-empty *interpretation domain* and $\cdot^\mathcal{I}$ the *interpretation function* that assigns an element or a set of elements in $\Delta^\mathcal{I}$ to a symbol of Σ. Each interpretation $\mathcal{I}$ that satisfies all knowledge base assertions in $\mathcal{T} \cup \mathcal{A}$ is called a *model*. The

set of all models of $\mathcal{K}$ is denoted by $Mod(\mathcal{K})$ and if $Mod(\mathcal{K}) \neq \emptyset$ we call $\mathcal{K}$ *satisfiable* or *consistent* [1, 4]. Otherwise $\mathcal{K}$ is called *inconsistent*. A knowledge base $\mathcal{K}$ is called *incoherent*, if there exists a concept C in Σ that is *unsatisfiable*, i.e., every model for $\mathcal{K}$ assigns the empty set to C. Roughly speaking, coherence considers the TBox of a knowledge base, whereas consistency addresses the ABox taking into account the consequences that follow from the TBox. Even though incoherence and inconsistency are strongly related and incoherences potentially cause inconsistencies (i.e., if there exist an individual of an unsatisfiable concept), there might exist knowledge bases that are incoherent but consistent [4]. $\mathcal{K} \models \phi$ denotes that $\mathcal{K}$ logically entails or satisfies a closed first-order logic sentence (formula) ϕ, if $\phi^{\mathcal{I}}$ is true for every $\mathcal{I} \in Mod(\mathcal{K})$. If a set of closed sentences denoted by F is entailed by $\mathcal{K}$, we can also write $\mathcal{K} \models F$ [19].

According to Kalyanpur at al. [8] an *explanation* (or justification) for $\mathcal{K} \models \phi$ is a subset $\mathcal{K}'$ of $\mathcal{K}$ such that $\mathcal{K}' \models \phi$ while $\mathcal{K}'' \not\models \phi$ for all $\mathcal{K}'' \subset \mathcal{K}'$. An explanation can be understood as a minimal reason that explains why ϕ follows from $\mathcal{K}$. Analogously, given an inconsistent knowledge base $\mathcal{K}$, we are especially interested in explanations for the inconsistency, i.e., minimal subsets $\mathcal{K}'$ of $\mathcal{K}$ such that $Mod(\mathcal{K}') = \emptyset$. With respect to our running example, $\mathcal{T} \cup \mathrm{DS}_A \cup \mathrm{DS}_B$ is an inconsistent knowledge base. An explanation for the inconsistency is the set

$$\{Man(\mathbf{Homer}), gaveBirthTo(\mathbf{Homer}, \mathbf{Lisa}),\\ Woman \sqsubseteq \neg Man, \exists gaveBirthTo \sqsubseteq Woman\}.$$

Further explanations do not exist within our example.

Note that we are only interested in the elements of the explanation that origin from the integrated data sources, since we understand the terminology $\mathcal{T}$ as a commonly accepted standard binding for all sources. By computing a *hitting set* $\mathcal{H}$ over all inconsistency explanations, reduced to the assertions that origin from one or some of the data sources, the inconsistency of the entire knowledge base can be resolved by removing the assertions in $\mathcal{H}$ from the respective sources.

Especially the process of identifying unsatisfiable concepts or inconsistencies, generating explanations for them and proposing some repair plans to resolve found contradictions is called *ontology debugging*.

2.2 Conjunctive Queries

Queries, especially *conjunctive queries* over the TBox $\mathcal{T}$ of a knowledge base $\mathcal{K}$, are in Datalog notation of the form $q(\boldsymbol{x}) \leftarrow conj(\boldsymbol{x}, \boldsymbol{y})$ where $\boldsymbol{x}$ are *distinguished variables* that are part of the *head* $q(\boldsymbol{x})$ of a query q whereas $\boldsymbol{y}$ are *non-distinguished variables* and do not occur in the head. If a variable does not correspond to the set of distinguished variables and does not occur in at least two query atoms, the variable is called *unbound* and is denoted by $\_$. Atoms like $B(x)$, $R(x,y)$, $x = y$ or $x \neq y$, in which x and y are either constants in Σ or variables in $\boldsymbol{x}$ or $\boldsymbol{y}$, and B a concept name or value-domain in $\mathcal{T}$ and R a role or attribute name in $\mathcal{T}$, can be *conjugated* and are denoted by $conj(\boldsymbol{x},\boldsymbol{y})$ that builds the *body* of q. *Unions of conjunctive queries* are denoted as $q(\boldsymbol{x}) \leftarrow conj_1(\boldsymbol{x}, \boldsymbol{y}_1)$,

$\ldots$, $q(\boldsymbol{x}) \leftarrow conj_n(\boldsymbol{x}, \boldsymbol{y}_n)$. With respect to the TBox $\mathcal{T}$ specified in our example we can select all Women that have descendants using the following query:

$$q(x) \leftarrow Woman(x), hasDescendant(x, \_)$$

Certain answers to $q(\boldsymbol{x})$ over $\mathcal{K}$ is the set of all tupels $\boldsymbol{t}$ of constants in $\mathcal{K}$ denoted by $answ(q, \mathcal{K})$ such that by substituting the variables $\boldsymbol{x}$ by $\boldsymbol{t}$ we have $\mathcal{K} \models q(\boldsymbol{t})$, i.e., for every interpretation $\mathcal{I} \in Mod(\mathcal{K})$ the condition $\boldsymbol{t}^{\mathcal{I}} \in \boldsymbol{q}^{\mathcal{I}}$ holds [2,18].

In Section 3, we will introduce the notion of a clash query. The answers to a clash query are those constants involved in inconsistencies, i.e., that appear in the assertions, which are elements of inconsistency explanations. Moreover, we show how to reconstruct these assertions from the given query and its answers.

2.3 *DL-Lite*$_{\mathcal{A}}$ and its Family

A specific lightweight family of DLs, called *DL-Lite* family, was proposed by Calvanese et al. [2] with the aim to find the trade-off between expressiveness and reasoning complexity. This specifically tailoring enables reasoning in PTIME in size of the TBox and query answering in LOGSPACE or rather in AC^0 in size of the ABox. Furthermore it has been shown that members of the *DL-Lite* family are one of the maximal logics that allow first-order logic (FOL)-rewritability of conjunctive query answering and therewith a processing of query answering through standard database technology (like in OBDA). Especially for that reason our approach is based on members of *DL-Lite*.

Based on *DL-Lite*$_{core}$ which builds the base line of the *DL-Lite* family, *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ are the simplest members extending *DL-Lite*$_{core}$. However, *DL-Lite*$_{\mathcal{A}}$ combines features of both with some restrictions and unlike *DL-Lite*$_{\mathcal{F}}$ and *DL-Lite*$_{\mathcal{R}}$ distinguishes between concepts from value-domains and roles from attributes (binary relations between concepts and value-domains). Due to that reason and with the aim of being especially suitable for dealing with and reasoning on large ABoxes we have selected *DL-Lite*$_{\mathcal{A}}$ for our approach. All features of those basic *DL-Lite* members and their distinction are shown in Table 1 by describing the syntax on which possible expressions are formed.

In Table 1 $\top_C$ denotes the *top* or *universal concept*, $\bot_C$ the *bottom* or *empty concept*, A an *atomic concept*, B a *basic concept* and C a *general concept*. Similar to that, we have *atomic roles* denoted by P, *basic roles* by Q and *general roles* by R. *Atomic attributes* are represented by U and *general attributes* by V whereas E denotes a *basic value-domain* and F a *value-domain expression*. Furthermore, as typical in DLs $\exists Q$ (*unqualified existential restrictions*) represent objects that are related by role Q to some objects, $\exists Q.C$ (*qualified existential restrictions*) denote objects that are related by Q to objects denoted by concept C, $\neg$ denotes the negation of concepts, roles or attributes and P^- is used to represent the inverse of role P. Concerning an attribute U its *domain* is denoted by $\delta(U)$ and its *range* (set of values) by $\rho(U)$. *Value domains* are represented by $T_1 \mid \ldots \mid T_n$, where each T_i denotes a pairwise disjoint data type of values and $\top_D$ the *universal value-domain*. [2,18]

Table 1. The *DL-Lite* Family

	DL-Lite$_{core}$	***DL-Lite***$_{\mathcal{F}}$	***DL-Lite***$_{\mathcal{R}}$	***DL-Lite***$_{\mathcal{A}}$
$\boldsymbol{B}$ ::=	$\bot_C \mid A \mid \exists Q$			$\bot_C \mid A \mid \exists Q \mid \delta(U)$
$\boldsymbol{C}$ ::=	$\top_C \mid B \mid \neg B$		$\top_C \mid B \mid \neg B \mid \exists Q.C$	
$\boldsymbol{Q}$ ::=	-	$P \mid P^-$		
$\boldsymbol{R}$ ::=	-		$Q \mid \neg Q$	
$\boldsymbol{E}$ ::=	-			$\rho(U)$
$\boldsymbol{F}$ ::=	-			$\top_D \mid T_1 \mid \ldots \mid T_n$
$\boldsymbol{V}$ ::=	-			$U \mid \neg U$
TBox assertions	$B \sqsubseteq C$	$B \sqsubseteq C$ (funct Q)	$B \sqsubseteq C$ $Q \sqsubseteq R$	$B \sqsubseteq C$ $Q \sqsubseteq R$ $E \sqsubseteq F$ $U \sqsubseteq V$ (funct Q)* (funct U)*
ABox assertions	$A(a)$ $P(a,b)$			$A(a)$ $P(a,b)$ $U(a,v)$

*iff Q (where Q is P or P^-) and U are primitive, which means that there exists no specialization.

Besides that, the TBox $\mathcal{T}$ and the ABox $\mathcal{A}$ are finite sets of assertions of the form given for each DL in Table 1. TBox assertions of the form $B \sqsubseteq C$ denotes *concept inclusions*, $Q \sqsubseteq R$ *role inclusion*, $E \sqsubseteq F$ *value-domain inclusion* and $U \sqsubseteq V$ *attribute inclusion*. *Functionality assertions* on roles and attributes in $\mathcal{T}$ are denoted by (funct Q) and (funct U), respectively. TBox assertions of the form $B_1 \sqsubseteq B_2$ and $Q_1 \sqsubseteq Q_2$ are called *positive inclusions* (PI) whereas $B_1 \sqsubseteq \neg B_2$ and $Q_1 \sqsubseteq \neg Q_2$ *negative inclusions* (NI). For ABox assertions a and b represent object constants and v represents a value constant.

Concerning the semantics of membership assertions in $\mathcal{A}$, the *DL-Lite* family imposes the *unique name assumption*, meaning that if the constants a and b are distinct then $a^{\mathcal{I}} \neq b^{\mathcal{I}}$. In terms of further semantics we refer to the more general definition we have given above and to the more precise definitions given in [2,18].

3 Inconsistency Detection

In this section we first explain different clash types that may occur in a *DL-Lite*$_{\mathcal{A}}$-based knowledge base. Basing on that, we define a translation function that is used in our approach to generate queries for inconsistency detection. Before describing and proving our algorithm for inconsistency detection and generation of its explanations we elucidate the previously defined clash queries for distributed environments.

3.1 Inconsistency Detection in *DL-Lite*$_{\mathcal{A}}$ Knowledge Bases

The consistency of a knowledge base can be determined by searching for obvious contradictions (also known as *clashes*) in the ABox. According to the work of

Lembo et al. [13], in a *DL-Lite*$_\mathcal{A}$ knowledge base clashes can be caused by only one of the following reasons:

(a) an instantiation of an unsatisfiable (incoherent) concept, role or attribute such that $\mathcal{T} \models A \sqsubseteq \neg A$ and $A(a) \in \mathcal{A}$ (respectively $\mathcal{T} \models P \sqsubseteq \neg P$ and $P(a, b) \in \mathcal{A}$ for roles, and $\mathcal{T} \models U \sqsubseteq \neg U$ and $U(a, v) \in \mathcal{A}$ for attributes)

(b) ABox assertions contradicting axioms that prohibit self-interrelation of individuals such that $\mathcal{T} \models P \sqsubseteq \neg P^-$ or $\mathcal{T} \models \exists P \sqsubseteq \neg \exists P^-$ and $P(a, a) \in \mathcal{A}$

(c) incorrect data types such that $\mathcal{T} \models \rho(U) \sqsubseteq T$, $U(a, v) \in \mathcal{A}$ and $v^\mathcal{I} \notin T^\mathcal{I}$

(d) ABox assertions contradicting negative inclusions such that for example $\mathcal{T} \models A \sqsubseteq \neg \exists P$ and $\{A(a), P(a, b)\} \subseteq \mathcal{A}$

(e) ABox assertions contradicting role functionality such that (funct P) $\in \mathcal{T}$ and $\{P(a, b_1), P(a, b_2)\} \subseteq \mathcal{A}$ (respectively (funct P^-) $\in \mathcal{T}$ and $\{P(a_1, b), P(a_2, b)\} \subseteq \mathcal{A}$ for the functionality of a inverse role)

(f) ABox assertions contradicting attribute functionality such that (funct U) $\in \mathcal{T}$ and $\{U(a, v_1), U(a, v_2)\} \subseteq \mathcal{A}$

Detection of such clashes requires that not only explicit but also implicit knowledge has to be taken into consideration. Since in our approach the focus is on distributed environments like in OBDI, implicit knowledge can be not only derived from the ontologies of each data source but also from the conceptual and centralized view. To obtain this complete knowledge especially by querying there exist two different ways to compute the certain answers.

One way is the materialization of ABoxes, where a materialized ABox is an original ABox extended by all assertions that can be additionally implied by the TBox(es) defined locally but also centrally. Queries will then be evaluated against the materialized ABox. This method, which is known as *forward-chaining* (or also bottom-up), requires the duplication of information. Like in data warehousing the redundant data have to be kept up-to-date on each modification and requires therewith additional resources. For that reasons such an approach is intractable in our application scenario.

Instead, we apply the method of *backward-chaining* (also known as top-down) where the ABoxes can be kept in the original state. The original query is reformulated (rewritten) with respect to the TBox to the effect that all knowledge relevant for the computation of the certain answers to that query is compiled into a set of rewritten queries (unions of conjunctive queries). Roughly speaking, if a query atom addresses individuals of a specific concept, the rewritten queries will contain atoms addressing all possible concepts, roles and attributes that also provide individuals of the originally requested concept. As already mentioned above, especially for *DL-Lite*$_\mathcal{A}$ such rewritings can be done in PTIME and query answering in AC^0 each in size of the TBox and ABox, respectively [11,18].

To utilize this feature of computing certain answers to a query by its rewriting our approach of inconsistency detection comprises the generation of so called *clash queries*.

3.2 Clash Query Generation

According to the clash definitions given above and based on the work of Calvanese et al. [2] we are able to define a translation function τ that generates queries for inconsistency detection from negative inclusions, functionality assertions or value-domain inclusions in $\mathcal{T}$, denoted by $\mathcal{T}_n$. If any of such clash queries delivers non-empty result sets, we can conclude that the delivered individuals cause inconsistencies with respect to elements in $\mathcal{T}_n$ addressed by the atoms of the generated query. For *DL-Lite*$_\mathcal{A}$, the translation function τ and all required kinds of generated clash queries are listed below in Datalog notation:

(i) $\tau(B_1 \sqsubseteq \neg B_2) = q(x) \leftarrow b_1, b_2$, where $b_i = A_i(x)$ if $B_i = A_i$, $b_i = P_i(x, \_)$ if $B_i = \exists P_i$, $b_i = P_i(\_, x)$ if $B_i = \exists P_i^-$, and $b_i = U_i(x, \_)$ if $B_i = \delta(U_i)$
(ii) $\tau(R_1 \sqsubseteq \neg R_2) = q(x, y) \leftarrow r_1, r_2$, where $r_i = P_i(x, y)$ if $R_i = P_i$, $r_i = P_i(y, x)$ if $R_i = P_i^-$, and $r_i = U_i(x, y)$ if $R_i = U_i$
(iii) $\tau(\rho(U) \sqsubseteq T_i) = q(x, y) \leftarrow U(x, y), T_i \neq \mathit{datatype}(y)$
(iv) $\tau((\mathsf{funct}\ R)) = q(x, y, z) \leftarrow r_1, r_2, y \neq z$, where $r_1 = P(x, y)$ if $R = P$, $r_1 = P(y, x)$ if $R = P^-$, $r_1 = U(x, y)$ if $R = U$, $r_2 = P(x, z)$ if $R = P$, $r_2 = P(z, x)$ if $R = P^-$, and $r_2 = U(x, z)$ if $R = U$

Obviously, the first two queries addresses the defined clash types (a), (b) and (d), the third query clash type (c) and the last two clash types (e) and (f). For our running example defined in Section 1 the following complete list of clash queries can be derived:

$$
\begin{aligned}
q(x) &\leftarrow \mathit{Man}(x), \mathit{Woman}(x)\\
q(x, y, z) &\leftarrow \mathsf{hasBirthday}(x, y), \mathsf{hasBirthday}(x, z), y \neq z\\
q(x, y) &\leftarrow \mathit{hasBirthday}(x, y), \texttt{xsd:dateTime} \neq \mathit{datatype}(y)\\
q(x, y, z) &\leftarrow \mathit{hasDNA}(x, y), \mathit{hasDNA}(x, z), y \neq z\\
q(q, y, z) &\leftarrow \mathit{hasDNA}(y, x), \mathit{hasDNA}(z, x), y \neq z\\
q(x, y) &\leftarrow \mathit{hasAncestor}(x, y), \mathit{hasAncestor}(y, x)\\
q(x, y) &\leftarrow \mathit{hasDescendant}(x, y), \mathit{hasDescendant}(y, x)
\end{aligned}
$$

Now we have to apply the rewriting techniques introduced above to each of these queries. Rewriting the first clash query results in the following Datalog program:

$$
\begin{aligned}
q(x) &\leftarrow \mathit{Woman}(x), \mathit{Man}(x)\\
q(x) &\leftarrow \mathit{gaveBirthTo}(x, \_), \mathit{Man}(x)
\end{aligned}
$$

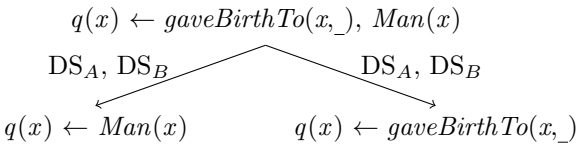
By identification of query parts that return some results due to an inconsistency in the knowledge base, it is possible to pinpoint those ABox assertions that are responsible for the inconsistency. In case of our example above, the last query part will return **Homer** and for that reason we know that {*gaveBirthTo*(**Homer**,_), *Man*(**Homer**)} is an explanation for the inconsistency. Since the translation function may produces query atoms containing unbound variables (denoted by _), the derived ABox assertions are not complete. This is the case especially for

existential restrictions in clash query type (i). For this special case a subsequent query to select the unbound values can be formulated. Another option is to expand this specific type of generated clash queries by new distinguished variables for each unbound variable.

3.3 Clash Query Federation

Considering the distributive environment, the generated clash queries will be evaluated through a simple federation algorithm for our first experiments. The processing of a query starts with its rewriting according to the method of backward-chaining, taking the semantics of the central ontology into account. In the more general case mappings to external ontologies are also taken into account within this step. This results in unions of conjunctive queries that are equivalent to the original query. Since a query, i.e., its atoms may address several data sources each query atom is sent to all sources and its results are federated.

Following our running example, the following federated queries will be generated for the rewritten query of the form $q(x) \leftarrow \mathit{gaveBirthTo}(x,\_),\ \mathit{Man}(x)$:



The results of these queries are {**Homer**, **Bart**} and {**Homer**, **Marge**}. Computing the intersection of these sets, the final result of the original clash query is the singleton {**Homer**}.

3.4 Generating Explanations

Algorithm 1 is based on similar Consistent algorithms proposed by Calvanese et al. [2,3] and summarizes our approach for computing all federated inconsistency explanations for a *DL-Lite*$_\mathcal{A}$-based knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$. We first iterate over $\mathcal{T}_n$, which is the set of all negative inclusions, functionality assertions and value-domain inclusions in $\mathcal{T}$. In Algorithm 1 $\mathcal{T}_n$ is set by the function Determine$\mathcal{T}_n$. For each element α in $\mathcal{T}_n$ we apply the translation function τ to generate the corresponding clash query. Since the semantics of *DL-Lite*$_\mathcal{A}$ does not contain specializations of elements in functionality assertions, we only have to rewrite clash queries for negative inclusions and value-domain inclusions according to the method of backward-chaining. Implementations of such rewriting algorithms are for example PerfectRef given by Calvanese et al. [2] or TreeWitness constituted by Kontchakov et al. [12], which is more efficient than PerfectRef. Both algorithms are part of the –ontop– framework[1] that is used within our implementation. For the experimental evaluation in Section 4 we used the TreeWitness.

[1] `http://ontop.inf.unibz.it`

This rewriting step results in unions of conjunctive queries (q^S). We execute each conjunctive query φ of q^S separately, as explained in the previous section. If the result of φ is not empty, we use the query result R^S and the query itself to transform all result tupels into a set of clashing ABox assertions $\mathcal{C}'$, that represents a set of all inconsistency explanations related to that query result (TransformIntoAssertions in Algorithm 1). We omit the TBox elements in these explanations since we assume that $\mathcal{T}$ is commonly accepted and kept constantly. We collect all explanations $\mathcal{C}'$ of each conjunctive query φ in a overall set $\mathcal{C}$ of all inconsistency explanations.

Since $\mathcal{T}_n$, the set of all negative inclusions, functionality assertions and value-domain inclusions in $\mathcal{T}$ is finite and the termination of Rewrite$(q, \mathcal{T})$ (such as PerfectRef or TreeWitness) is assumed to be already established, the termination of this algorithm is given.

Algorithm 1: InconsistencyDetection$(\mathcal{K})$

Input: *DL-Lite*$_\mathcal{A}$ knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$
Output: all inconsistency explanations $\mathcal{C}$
begin
  $\mathcal{C} \leftarrow \emptyset$;
  $\mathcal{T}_n \leftarrow$ Determine$\mathcal{T}_n(\mathcal{T})$;
  foreach $\alpha \in \mathcal{T}_n$ **do**
    $q \leftarrow \tau(\alpha)$;
    $q^S \leftarrow \emptyset$;
    if α *is a negative inclusion or a value-domain inclusion* **then**
      $q^S \leftarrow$ Rewrite$(q, \mathcal{T})$;
    else
      $q^S \leftarrow \{q\}$;
    foreach $\varphi \in q^S$ **do**
      $R^S \leftarrow$ Answ$(\varphi, \mathcal{A})$;
      if $R^S \neq \emptyset$ **then**
        $\mathcal{C}' \leftarrow$ TransformIntoAssertions(R^S, φ);
        $\mathcal{C} \leftarrow \mathcal{C} \cup \mathcal{C}'$;
  return $\mathcal{C}$;
end

Proposition 1. *Let $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ be a DL-Lite$_\mathcal{A}$ knowledge base, where $\mathcal{A}$ is the union of distributed data sources. Then* InconsistencyDetection*$(\mathcal{K})$ generates the set of all inconsistency explanations for $\mathcal{K}$.*

Proof. Based on the work of Lembo et al. [13] *DL-Lite*$_\mathcal{A}$ knowledge bases may only contain those clash types that are listed in Section 3.1. We proceed by considering all these possible cases:

- To detect clashes of type (a), a corresponding clash query q must be able to address basic concepts (A, $\exists Q$, $\delta(U)$), basic roles (P, P^-) and atomic attributes (U) according to negative inclusions α explicitly defined in $\mathcal{T}$ appropriate to the syntax of *DL-Lite*$_\mathcal{A}$ (see Table 1). All these incidents are satisfied by clash queries of type (i) for basic concepts and of type (ii) for basic roles and atomic attributes. Furthermore, according to the work of Calvanese et al. [2] and Kontchakov et al. [12] $answ(q, \mathcal{K}) = answ(\mathsf{Rewrite}(q, \mathcal{T}), \mathcal{A})$, we consider not only $\alpha \in \mathcal{T}$ but also $\mathcal{T} \models \alpha$. Therefore the claim holds for clashes of type (a).
- To detect clashes of type (b), a corresponding clash query q must be able to address basic roles (P, P^-) and existential restrictions ($\exists Q$) according to negative inclusions α explicitly defined in $\mathcal{T}$ appropriate to the syntax of *DL-Lite*$_\mathcal{A}$. All these incidents are satisfied by clash queries of type (ii) for basic roles and of type (i) for existential restrictions. Furthermore, exactly like for clash type (a), we consider not only $\alpha \in \mathcal{T}$ but also $\mathcal{T} \models \alpha$. Therefore the claim holds for clashes of type (b).
- To detect clashes of type (c), a corresponding clash query q must be able to address atomic attributes (U) and value-domains (F) according to value-domain inclusions α explicitly defined in $\mathcal{T}$ appropriate to the syntax of *DL-Lite*$_\mathcal{A}$, since by definition of Poggi et al. [18] value-domains are pairwise disjoint. This is satisfied by clash queries of type (iii). Due to the fact that according to the *DL-Lite*$_\mathcal{A}$ syntax attributes can be specialized, we consider not only $\alpha \in \mathcal{T}$ but also $\mathcal{T} \models \alpha$. Therefore the claim holds for clashes of type (c).
- To detect clashes of type (d), the same holds as for clash type (a).
- To detect clashes of type (e), a corresponding clash query q must be able to address atomic roles (P, P^-) according to negative inclusions α explicitly defined in $\mathcal{T}$ appropriate to the syntax of *DL-Lite*$_\mathcal{A}$. All these incidents are satisfied by clash queries of type (iv) for atomic roles and atomic inverse roles. Since functional roles have to be primitive it is sufficient to consider only $\alpha \in \mathcal{T}$. Therefore the claim holds for clashes of type (e).
- To detect clashes of type (f), a corresponding clash query q must be able to address atomic attributes (U) according to negative inclusions α explicitly defined in $\mathcal{T}$ appropriate to the syntax of *DL-Lite*$_\mathcal{A}$. This is satisfied by clash queries of type (iv). Since functional attributes also have to be primitive it is sufficient to consider only $\alpha \in \mathcal{T}$. Therefore the claim holds for type (f). □

Repairing the detected inconsistencies, i.e., deciding which assertions should be eliminated, is beyond the scope of this paper. Several approaches have been already proposed to solve this problem [6]. Depending on the specifics of the setting one might, for example, be interested to remove a minimum number of assertions causing inconsistencies by computing a smallest minimal hitting set over all explanations. In most of these approaches it is required to have access to the set of all inconsistency explanations, which are generated by our approach.

4 Experimental Evaluation

In order to evaluate the performance of our approach we compare our implementation of algorithm InconsistencyDetection, called ClashSniffer, to the reasoning system Pellet [22]. Pellet offers a specific service for computing inconsistency explanation and can thus be directly compared to our approach. Moreover, we are conducting experiments with the Black-Box algorithm for computing explanations, which is implemented as component in the OWL API, using HermiT [7] as underlying reasoner. We have artificially generated some RDF datasets comprising 500, 5000, 10000, 50000 and 100000 ABox assertions according to the TBox definition of our running example. The collection of datasets is available at `http://www.researchgate.net/publication/263051841_ClashSniffer_Evaluation_Datasets`. Each dataset contains some assertions that will cause inconsistencies and are generated randomly with a rate about 2% of the complete number of assertions. All possible clash types given in Section 3.1 may occur within these datasets. Since the OWL 2 QL profile[2] is based on DL-Lite, we use it as specification language of our defined TBox.

Pellet and HermiT can only be applied to the non-distributed version of the dataset. For that reason we run our algorithm both in a local setting using a central repository that contains the complete dataset (ClashSniffer$_L$), and in a distributed environment (ClashSniffer$_F$). In the distributed environment the ABox assertions are randomly distributed over four data sources, represented by instances of Virtuoso (Open-Source Edition)[3]. In this setting we sent each query atom to all data sources and federated the results.

The results of our experimental evaluation are depicted in Table 2. It illustrates that first of all, the runtimes for the local and the distributed settings of our algorithm differ significantly. This is caused by the latency in the network and also by the fashion how the federated queries are executed. Since in our implementation we use ARQ, a query engine for Apache Jena, for each tupel that is returned as an answer for a query atom of a federated query, a new subquery for the next query atom that is related to the first one will be generated for all assigned data sources by default. Its also interesting to see that the runtimes for the distributed settings increase linear with respect to the problem size. This is not the case for the local setting, where the size of the ABox has only a minor impact on the overall runtime.

A surprising result is related to the performance of Pellet and HermiT. For the smallest dataset Pellet requires significantly more time to compute all explanations than our algorithm, in both the local and the distributed setting. Pellet takes more than five hours to compute explanations for the data set comprising 5000 assertions. We stopped the experiment after five hours. Contrary to Pellet, HermiT ends up on 500 assertions with an `OutOfMemoryError` despite of an assigned memory of 4GB. We are currently missing an appropriate explanation for this behaviour.

[2] `http://www.w3.org/TR/owl2-profiles`
[3] `http://virtuoso.openlinksw.com/dataspace/doc/dav/wiki/Main/`

Table 2. Experimental Evaluation Results

ABox Size		500	5,000	10,000	50,000	100,000
Pellet	**Execution (ms)**	1,713,901	>18,000,000	–	–	–
	♯Explanations	20 (13)	–	–	–	–
HermiT	**Execution (ms)**	[Error]	–	–	–	–
	♯Explanations	–	–	–	–	–
Clash Sniffer$_L$	**Execution (ms)**	359	563	718	1,313	2,171
	♯Explanations	13	113	191	1,050	2,121
Clash Sniffer$_F$	**Execution (ms)**	29,360	234,878	464,973	2,227,466	4,541,140
	♯Explanations	13	113	191	1,050	2,121

Comparing the generated inconsistency explanations of Pellet to the ones that are produced by our algorithm, it can be observed that both approaches detected the same explanations. However, Pellet generates (particularly concerning inconsistencies on attributes) explanation sets that are not minimal, i.e., some of the generated sets are supersets of (minimal) explanations. Since supersets of the same explanation are computed in some cases, Pellet produces a higher number of explanations. We have manually analysed the explanations generated by Pellet for the test case with 500 assertions. After mapping each superset of an explanation on the minimal explanation that was contained in the superset, the results for Pellet and our approach were the same. Restricting the number of explanations to small numbers (e.g., 5 or 10) the Black-Box approach is also capable of generating an output. However, again, the generated sets are often proper supersets of an explanation and the algorithm cannot generate such sets if we increase the number of requested explanations. Moreover, our algorithm generates in both, the local and the distributed setting, the same explanations, which is in line with our theoretical considerations.

5 Related Work

State-of-the-art DL or OWL reasoners that are used for inconsistency detection and its explanations basically process local knowledge bases and are therefore inappropriate for distributed environments. Moreover, regardless of the supported language expressiveness or the underlying reasoning method like widely used tableau algorithms as in FaCT++ [23], Pellet [22], or RacerPro [5], the hypertableau technique of HermiT [7,16], consequence-driven approaches like the ones described by Kazakov [9] or Simančík et al. [21], or methods described by Motik & Sattler [15] or Kazakov & Motik [10] that are based on resolution.

To the best of our knowledge there is currently no ready to operate approach that addresses inconsistency detection in the context of distributed knowledge bases, like in OBDI. However, there are some works running in a similar direction.

Calvanese et al. [2] present apart from the initial definition of the *DL-Lite* family among others a definition of a translation function δ that transforms negative inclusions and functionality assertions into queries (FOL formulas). This translation function is applied in the algorithm Consistent to each negative inclusions and functionality assertions that can be logically implied from the given knowledge base (TBox). Afterwards, a Boolean query comprising the union of all queries generated by δ is evaluated over the given ABox. An implementation of this proposed solution is given by the –ontop– framework, already mentioned above. In contrast to our approach the work of Calvanese et al. do not support *DL-Lite*$_{\mathcal{A}}$ knowledge bases. Beside this, Calvanese et al. [3] expand their approach to *DLR-Lite*$_{\mathcal{A},\sqcap}$, a new member of the *DL-Lite* family that permits among others the use of n-ary relations and conjunctions on the left-hand side of inclusion assertions. Despite the fact that the algorithms Consistent proposed in these works are similar to our algorithm, both only identify if there are some inconsistencies but do not specify these inconsistencies in greater detail or give some explanations to them. Furthermore, our approach additionally comprises the federation of distributed *DL-Lite* knowledge bases.

The approach proposed by Lembo et al. [13, 14] facilitate meaningful query results over inconsistent *DL-Lite* knowledge bases under different inconsistency-tolerant semantics. Therefor, an additional rewriting under the defined semantics is applied to the rewritings produced by PerfectRef in order to implement inconsistency tolerance on query answering. Roughly speaking, queries generated by applying backward-chaining are extended in such a way that triples producing inconsistencies will not be considered on query answering. For this purpose, similar to our approach ontology axioms that can be contradicted by ABox assertions are used for query generation, i.e., its extension, but with the difference that their aim is to exclude all assertions that cause inconsistencies from query evaluation whereas our claim is to select these assertions, which is exactly the opposite. Although the method of Lembo et al. is suitable for accessing distributed data, such as in OBDI, it is not designed for inconsistency detection and explanation.

Serafini & Tamilin [20] offer a tableau-based DL reasoning algorithm for pairwise interrelated standalone repositories. This approach enables distributed reasoning capabilities by following principles of peer-to-peer networks. Since each integrated data source have to implement a peer ontology manager and to provide local and global reasoning services, the imposed requirements restrict the integration of data sources and are contrary to principles followed in OBDI.

6 Conclusions and Future Work

In this paper we have described an approach of efficient inconsistency detection in distributed knowledge bases based on *DL-Lite*$_{\mathcal{A}}$. The described approach relies on the generation of clash queries that are evaluated over all integrated data sources. We have further depicted an algorithm that detects existing inconsis-

tencies and generates explanations to them. Beside a proof of completeness we have also shown experimental evaluation results for our algorithm.

Since in this paper we cover only one part of ontology debugging, namely the identification of inconsistencies and its explanations, in our future work we will address the generation of extended explanations, e.g., comprising the corresponding data sources, and the task of proposing some repair plans. Furthermore, we will evaluate our approach by using "real" instead of artificially generated data sets. In addition we plan to compare our approach of inconsistency detection also against some *DL-Lite* tailored solutions, such as the OBDA management system Mastro[4]. Due to the fact that we have used just a simple federation algorithm we will integrate the proposed approach into the federation engine ELITE [17]. ELITE was developed with the purposes of efficient and complete processing of federated queries in distributed environments. Especially the use of the R-Tree-based index of ELITE guarantee that only those query parts are evaluated that probably deliver some results. By this means the task of inconsistency detection in distributed environments can be solved more efficient.

Acknowledgments. This research has been partially supported by the SEMANCO project (`http://www.semanco-project.eu`), which is being carried out with the support of the Seventh Framework Programme "ICT for Energy Systems" 2011–2014, under the grant agreement no. 287534.

References

1. Baader, F.: The description logic handbook: theory, implementation, and applications. Cambridge: Cambridge University Press (2003)
2. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Tractable reasoning and efficient query answering in description logics: The DL-Lite family. Journal of Automated Reasoning 39(3), 385–429 (2007)
3. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Rosati, R.: Data complexity of query answering in description logics. Artificial Intelligence 195, 335–360 (2013)
4. Flouris, G., Huang, Z., Pan, J.Z., Plexousakis, D., Wache, H.: Inconsistencies, negations and changes in ontologies. In: Proceedings of the National Conference on Artificial Intelligence. vol. 21, p. 1295. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999 (2006)
5. Haarslev, V., Müller, R.: RACER system description. In: Automated Reasoning, pp. 701–705. Springer (2001)
6. Haase, P., Qi, G.: An analysis of approaches to resolving inconsistencies in DL-based ontologies. In: Proceedings of the International Workshop on Ontology Dynamics (IWOD-07). pp. 97–109 (2007)
7. Horrocks, I., Motik, B., Wang, Z.: The HermiT OWL Reasoner. In: Proceedings of the 1st International Workshop on OWL Reasoner Evaluation (ORE-2012), Manchester, UK (2012)

[4] `http://www.dis.uniroma1.it/~mastro`

8. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of OWL DL entailments. In: The Semantic Web. pp. 267–280. Springer (2007)
9. Kazakov, Y.: Consequence-driven reasoning for Horn $\mathcal{SHIQ}$ ontologies. In: IJCAI. vol. 9, pp. 2040–2045 (2009)
10. Kazakov, Y., Motik, B.: A Resolution-Based Decision Procedure for $\mathcal{SHOIQ}$. Journal of Automated Reasoning 40(2-3), 89–116 (2008)
11. Kiryakov, A., Damova, M.: Storing the Semantic Web: Repositories. Handbook of Semantic Web Technologies 1, 231–297 (2011)
12. Kontchakov, R., Lutz, C., Toman, D., Wolter, F., Zakharyaschev, M.: The combined approach to ontology-based data access. In: Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Three. pp. 2656–2661. AAAI Press (2011)
13. Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., Savo, D.F.: Query rewriting for inconsistent DL-Lite ontologies. In: Web Reasoning and Rule Systems, pp. 155–169. Springer (2011)
14. Lembo, D., Lenzerini, M., Rosati, R., Ruzzi, M., Savo, D.F.: Inconsistency-Tolerant First-Order Rewritability of DL-Lite with Identification and Denial Assertions. In: Proceedings of the 25th International Workshop on Description Logics (2012)
15. Motik, B., Sattler, U.: A comparison of reasoning techniques for querying large description logic aboxes. In: Logic for programming, artificial intelligence, and reasoning. pp. 227–241. Springer (2006)
16. Motik, B., Shearer, R., Horrocks, I.: Hypertableau reasoning for description logics. Journal of Artificial Intelligence Research 36(1), 165–228 (2009)
17. Nolle, A., Nemirovski, G.: ELITE: An Entailment-based Federated Query Engine for Complete and Transparent Semantic Data Integration. In: Proceedings of the 26th International Workshop on Description Logics (2013)
18. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Journal on data semantics X, pp. 133–173. Springer (2008)
19. Rudolph, S.: Foundations of description logics. In: Reasoning Web. Semantic Technologies for the Web of Data, pp. 76–136. Springer (2011)
20. Serafini, L., Tamilin, A.: Local tableaux for reasoning in distributed description logics. In: Proceedings of the 17th International Workshop on Description Logics. vol. 4, pp. 100–109 (2004)
21. Simančík, F., Kazakov, Y., Horrocks, I.: Consequence-based reasoning beyond Horn ontologies. In: Proceedings of the Twenty-Second international joint conference on Artificial Intelligence-Volume Volume Two. pp. 1093–1098. AAAI Press (2011)
22. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical owl-dl reasoner. Web Semantics: science, services and agents on the World Wide Web 5(2), 51–53 (2007)
23. Tsarkov, D., Horrocks, I.: FaCT++ description logic reasoner: System description. In: Automated reasoning, pp. 292–297. Springer (2006)
24. Wache, H., Voegele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., Hübner, S.: Ontology-based integration of information - a survey of existing approaches. In: IJCAI-01 workshop: ontologies and information sharing. vol. 2001, pp. 108–117. Citeseer (2001)