Fast ABox Consistency Checking using Incomplete Reasoning and Caching

Christian Meilicke[1], Daniel Ruffinelli[1],
Andreas Nolle[2], Heiko Paulheim[1], and Heiner Stuckenschmidt[1]

[1] Research Group Data and Web Science
University of Mannheim, Germany
`christian|daniel|heiko|heiner@informatik.uni-mannheim.de`
[2] Data Science, Department of Business and Computer Science
Albstadt-Sigmaringen University, Germany
`nolle@hs-albsig.de`

Abstract. Reasoning with complex ontologies can be a resource-intensive task, which can be an obstacle, e.g., for real-time applications. Hence, weakening the constraints of soundness and/or completeness is often an approach to practical solutions. In this paper, we propose an extension of incomplete reasoning methods for checking the consistency of a large number of ABoxes against a given TBox. In particular, we use and extend the clash queries proposed by Lembo et al. [10] for *DL-Lite* to compute inconsistent patterns of ABox assertions. By caching instantiations of these patterns, we are able to reduce the amount of reasoning required to determine the inconsistency of an ABox with every previously processed ABox. We present experimental results of our approach in terms of runtime and accuracy and compare it against complete reasoning techniques, the reasoning approach for *DL-Lite*$_{\mathcal{A}}$, and an approximate reasoning approach based on machine learning proposed in [16].

1 Introduction

Ontologies, and the reasoning with ontologies, are well established techniques for capturing and processing knowledge. In the past decades, a large body of research has been conducted on optimizing reasoning systems for ontology languages of different expressiveness. So far, the major part of research on ontology reasoning focuses on developing reasoning systems that are both sound and complete. However, as already argued in [16], reasoning results that are 100% accurate are not required in many use cases for which ontology reasoning has been proposed and/or applied in the past, e.g., information retrieval, recommender systems, or activity recognition. On the other hand, many of those use cases have very strict performance requirements, as they are usually applied in real time settings.

These considerations led to the development of reasoning systems that weaken the constraint of soundness or completeness for the sake of better performance. In this paper, we propose an approach for checking the consistency of many ABoxes against the same TBox, a task to which many real world reasoning tasks can be reduced. Our approach is sound but not complete for detecting inconsistencies in OWL 2 ontologies.

It builds upon *DL-Lite*$_\mathcal{A}$ clash queries [11] and extends them in order to detect inconsistencies beyond the scope of *DL-Lite*$_\mathcal{A}$. Furthermore, we propose a caching method to avoid costly calls to a reasoner, which becomes more effective with every processed ABox.

We conduct comprehensive experiments on two real life datasets and compare our method against three different types of reasoning approaches: First, we apply complete reasoning techniques using HermiT [7]. Second, we apply the reasoning approach for inconsistency detection in *DL-Lite*$_\mathcal{A}$ as proposed by Lembo et al. [10]. Third, we have reimplemented the approximate reasoning approach based on machine learning, proposed by Paulheim and Stuckenschmidt [16]. The results indicate that our approach is highly efficient and capable of detecting significantly more inconsistencies than other incomplete methods. We analyze the results of our experiments and explain under which conditions a method that is based on caching (parts of) explanations is a better choice than a method that is based on learning from previously seen examples and vice versa.

The rest of this paper is structured as follows. Section 2 introduces some basic concepts, the preliminaries on checking consistency in *DL-Lite*$_\mathcal{A}$, and the basic idea of using machine learning techniques as proposed in [16] to solve the given problem. Section 3 introduces our approach, which is based on the extension of the techniques proposed for *DL-Lite*$_\mathcal{A}$ combined with a caching technique. Section 4 discusses experimental results both w.r.t. result quality and runtime performance. We conclude with a summary and an outlook on future work.

2 Preliminaries and Related Work

Within this section we first recall the notion of inconsistency and explanation. We also argue why explanations are useful for our problem (Section 2.1). Then we introduce the description logics *DL-Lite*$_\mathcal{A}$ (Section 2.2) before we finally explain how inconsistencies can be detected in *DL-Lite*$_\mathcal{A}$ (Section 2.3). Later on we use the *DL-Lite*$_\mathcal{A}$ query expansion techniques as a baseline in our experiments and extend this approach to detect inconsistencies within ontologies that are beyond *DL-Lite*$_\mathcal{A}$. In Section 2.4 we present and discuss the idea of using machine learning for detecting inconsistencies.

2.1 Inconsistencies and Explanations

In the following we use $\mathcal{T}$ to refer to a TBox that defines the vocabulary used in a set of ABoxes $\mathcal{A}_1$ to $\mathcal{A}_n$. In description logics, an interpretation $\mathcal{I}$ that satisfies all axioms in $\mathcal{T}$ and all assertions in $\mathcal{A}$ is called a model of $\mathcal{T} \cup \mathcal{A}$. If such a model exists, $\mathcal{T} \cup \mathcal{A}$ is called consistent. Otherwise, $\mathcal{T} \cup \mathcal{A}$ is called inconsistent [1, 3]. The inconsistency of an ontology is usually a sign for an error, i.e., a sign for a faulty axiom or assertions. In our setting we assume that the TBox $\mathcal{T}$ is not causing the problem, but helps to reveal a mistake in (at least) one of the assertions in an ABox.

According to Kalyanpur et al. [8], an *explanation* (or justification) for an assertion or an axiom ϕ is a subset $\mathcal{O}'$ of $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$ such that $\mathcal{O}' \models \phi$ while $\mathcal{O}'' \not\models \phi$ for all $\mathcal{O}'' \subset \mathcal{O}'$. An explanation can be understood as a minimal reason that explains why ϕ follows from $\mathcal{O}$. Analogously, given an inconsistent ontology $\mathcal{O}$, we are interested in explanations for the inconsistency, i.e., minimal subsets $\mathcal{O}'$ of $\mathcal{O}$ such that there exists no model for $\mathcal{O}'$. More precisely, a minimal inconsistent subset $\mathcal{O}'$ (also referred to as MIS) is a subset of $\mathcal{O}$ such that $\mathcal{O}'$ is inconsistent while $\mathcal{O}''$ is consistent for all $\mathcal{O}'' \subset \mathcal{O}'$. An example for a MIS is shown as Example 1.

Example 1. This example shows a simplified inconsistency explanation for one of the ABoxes from the experiments with DBpedia and DOLCE. The inconsistency is related to the fact that *clintonMorrison11* is implicitly typed as a *Person*, but at the same time as a time span in the life of a person via the concept *Situation*.

$$\begin{aligned}
team(clintonMorrison11, irelandFootballTeam) && (1)\\
PhysicalObject \sqsubseteq \neg SocialObject && (2)\\
PhysicalAgent \sqsubseteq PhysicalObject && (3)\\
Person \sqsubseteq PhysicalAgent && (4)\\
Situation(clintonMorrison11) && (5)\\
\exists team \sqsubseteq Athlete && (6)\\
Athlete \sqsubseteq Person && (7)\\
Situation \sqsubseteq SocialObject && (8)
\end{aligned}$$

The example shows that the explanations for an inconsistency are not trivial and that mistakes in the ABox can only be detected via chains of relevant axioms. With respect to our setting, we are only interested in the ABox elements, since we trust in the correctness of the TBox axioms. For that reason, the relevant assertions are (1) and (5). It is important to understand that we can replace the concrete instances in (1) and (5) by any other pair of instances. This means that any instantiation of $team(x, y) \wedge Situation(x)$ will be inconsistent. The computation and caching of the relevant information that corresponds to such a partial explanation will be an important element of our approach. In particular, we focus only on a specific type of partial explanations, which correspond to the clash types in *DL-Lite*$_{\mathcal{A}}$.

2.2 *DL-Lite*$_{\mathcal{A}}$

DL-Lite is a family of lightweight description logics proposed by Calvanese et al. [2] with the aim to find a trade-off between expressiveness and reasoning complexity. This resulted in a family of languages where terminological reasoning can be done in PTIME in the size of the TBox and query answering in AC^0 in the size of the ABox. In *DL-Lite*$_{\mathcal{A}}$, which is a concrete member of the *DL-Lite* family, concept, role, value-domain, and attribute expressions are formed according to the following syntax:

$$\begin{aligned}
B &::= \bot_C \mid A \mid \exists Q \mid \delta(U) & E &::= \rho(U)\\
C &::= \top_C \mid B \mid \neg B \mid \exists Q.C & F &::= \top_D \mid T_1 \mid \ldots \mid T_n\\
Q &::= P \mid P^- & V &::= U \mid \neg U\\
R &::= Q \mid \neg Q
\end{aligned}$$

where $\top_C$ 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, $\exists Q$ (*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, 17]. In *DL-Lite*$_\mathcal{A}$ a knowledge base $\mathcal{K} = \langle \mathcal{T}, \mathcal{A} \rangle$ consists of a TBox $\mathcal{T}$ also known as schema, and an ABox $\mathcal{A}$, the extensional knowledge part which represents a data source.

The defined expressions can be used in TBox axioms in the following way. Axioms of the form $B \sqsubseteq C$ denote *concept inclusions*, $Q \sqsubseteq R$ *role inclusions*, $E \sqsubseteq F$ *value-domain inclusions* and $U \sqsubseteq V$ *attribute inclusions*. *Functionality assertions* on roles and attributes in $\mathcal{T}$ are denoted by $\mathsf{funct}\ Q$ and $\mathsf{funct}\ U$. TBox axioms 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. We refer the reader to [2, 17] for a discussion of the semantics of *DL-Lite*$_\mathcal{A}$, which we omit here due to the lack of space.

An example of an axiom that is not within the scope of *DL-Lite* is an axiom of the form $\exists partOf.Event \sqsubseteq Event$. If such an axiom is part of a MIS, the approach based on clash query expansion, which is shortly presented in the following section, will not be able to detect the respective inconsistency.

2.3 Inconsistency Detection in *DL-Lite*$_\mathcal{A}$

Lembo et al. [10] identified a collection of six different patterns that cause clashes in *DL-Lite*$_\mathcal{A}$ knowledge bases listed as follows. This collection is complete for *DL-Lite*$_\mathcal{A}$. This means that any inconsistency in $\mathcal{O} = \mathcal{T} \cup \mathcal{A}$, as long as the axioms and assertions are within the *DL-Lite*$_\mathcal{A}$ profile, can be detected by checking the following patterns. With respect to the following listing let a, b and c be individuals, let A and A' be named concepts, P and P' be roles, and let U be an attribute in accordance with the naming conventions of the previous section.

1) Instantiation of an unsatisfiable named concept, role, attribute
   a) $\mathcal{T} \models A \sqsubseteq \neg A$ and $A(a) \in \mathcal{A}$
   b) $\mathcal{T} \models P \sqsubseteq \neg P$ and $P(a, b) \in \mathcal{A}$
   c) $\mathcal{T} \models U \sqsubseteq \neg U$ and $U(a, v) \in \mathcal{A}$
2) Assertions contradicting axioms that prohibit self-interrelations
   a) $\mathcal{T} \models P \sqsubseteq \neg P^-$ and $P(a, a) \in \mathcal{A}$
3) Incorrect data types
   a) $\mathcal{T} \models \rho(U) \sqsubseteq T$ and $U(a, v) \in \mathcal{A}$ and $v^\mathcal{I} \notin T^\mathcal{I}$
4) Assertions contradicting negative inclusions

   a) $\mathcal{T} \models A \sqsubseteq \neg A'$ and $A(a), A'(a) \in \mathcal{A}$
   b) $\mathcal{T} \models A \sqsubseteq \neg \exists P$ and $A(a), P(a,b) \in \mathcal{A}$
   c) $\mathcal{T} \models A \sqsubseteq \neg \exists U$ and $A(a), U(a,v) \in \mathcal{A}$
   d) $\mathcal{T} \models A \sqsubseteq \neg \exists P^-$ and $A(b), P(a,b) \in \mathcal{A}$
   e) $\mathcal{T} \models P \sqsubseteq \neg P'$ and $P(a,b), P'(a,b) \in \mathcal{A}$
   f) $\mathcal{T} \models \exists P \sqsubseteq \neg \exists P'$ and $P(a,b), P'(a,c) \in \mathcal{A}$
   g) $\mathcal{T} \models \exists P \sqsubseteq \neg \exists P'^-$ and $P(a,b), P'(c,a) \in \mathcal{A}$
   h) $\mathcal{T} \models \exists P^- \sqsubseteq \neg \exists P'^-$ and $P(a,b), P'(c,b) \in \mathcal{A}$
5) Assertions contradicting role functionality
   a) $(\mathsf{funct}\ P) \in \mathcal{T}$ and $P(a,b), P(a,c) \in \mathcal{A}$ and $b \neq c$
   b) $(\mathsf{funct}\ P^-) \in \mathcal{T}$ and $P(a,c), P(b,c) \in \mathcal{A}$ and $a \neq b$
6) ABox assertions contradicting attribute functionality
   a) $(\mathsf{funct}\ U) \in \mathcal{T}$ and $U(a,v_1), P(a,v_2) \in \mathcal{A}$ and $v_1 \neq v_2$

In the context of *DL-Lite*$_{\mathcal{A}}$ it is sufficient to implement the $\models$ operator in terms of the *DL-Lite*$_{\mathcal{A}}$ expansion rules. All clashes related to clash type 4)a) can, for example, be detected by applying the *DL-Lite*$_{\mathcal{A}}$ expansion rules recursively on each directly stated disjointness axiom $B \sqsubseteq \neg C$. As a result, all relevant clashes of the type $\mathcal{T} \models A \sqsubseteq \neg A'$ are collected and the ABox can be checked against these inconsistency patterns. If we apply the approach on the axiom $\mathit{PhysicalObject} \sqsubseteq \neg \mathit{SocialObject}$ given a TBox that contains amongst others all of the axioms listed in in Example 1, the expansion rules will entail $\mathit{Situation} \sqsubseteq \neg \exists \mathit{team}$. This means that every ABox is inconsistent that instantiates $\mathit{team}(x,y) \wedge \mathit{Situation}(x)$.

2.4 Learning vs Computing Explanations

It has been a trend over the last years to train statistical models on large knowledge graphs in order to predict new facts about the world. An overview is given in [13]. These works are mainly concerned with the prediction of a fact that cannot be entailed by deductive reasoning. Opposed to that, in [16] the authors propose to mimic a reasoner for checking consistency by training a machine learning model. To this end, the authors propose to translate an ABox to a binary feature representation. On top of this representation a TBox specific classifier is learned that is able to distinguish between consistent and inconsistent ABoxes. The approach requires the usage of a reasoner to annotate the training examples, which are ABoxes translated to the feature representation, as consistent or inconsistent. Once the classifier has been trained, it mimics the behavior of the reasoner. Results presented in [16] have shown that the approach is highly efficient once the training phase has been finished.

A crucial aspect of the method is the chosen feature representation. In [16] the authors propose the use of path kernels introduced in [12] for generating the features. Without recalling the details, we point out that the generated features for Example 1 contain a feature for $\mathit{team}(x,y)$, a feature for $\mathit{Situation}(z)$, and another feature for the conjunction $\mathit{team}(x,y) \wedge \mathit{Situation}(x)$. This means that the classifier can also learn that each instantiation of $\mathit{team}(x,y) \wedge \mathit{Situation}(x)$ is inconsistent. In order to successfully learn that this pattern causes inconsistency, the training examples have to cover at least one inconsistent ABox $\mathcal{A}^*$ that makes use of this pattern. Moreover, there need to be some consistent training examples that use $team(x, y)$ only and some that use $Situation(x)$ only without the conjunction to avoid over-fitting. For the same purpose, the training examples must also cover consistent ABoxes that make use of the other concepts and roles in $\mathcal{A}^*$ that are not causing the inconsistency.

Contrary to an approach based on machine learning, we compute a partial explanation for the inconsistency of $\mathcal{A}^*$. By projecting the explanatory entailment to its corresponding assertional pattern, we are able to directly achieve the goal of the learning process without the need for annotating a sufficient number of samples with a standard reasoner. However, the computation of an explanation is known to be rather costly and several approaches have been proposed for this purpose [6, 8]. We base our work on the *DL-Lite* clash patterns. We will argue in the following section that we can use the *DL-Lite* techniques or a standard reasoner to check for $\mathcal{A}^*$ if a certain pattern, which might be part of a complex explanation, results in an inconsistency. By storing inconsistent patterns we can directly decide that each ABox that instantiates this pattern is inconsistent.

3 Our Approach

We describe two approaches for checking a sequence of ABoxes $\mathcal{A}_1, \ldots, \mathcal{A}_n$ against a given TBox $\mathcal{T}$. The first approach (Section 3.1) is a straightforward application of the inconsistency reasoning techniques of *DL-Lite*$_\mathcal{A}$ that are based on the clash types of Lembo et al [10]. Our approach (Section 3.2) extends and modifies this approach by using a reasoner that is fully compliant with the OWL 2 semantics.

3.1 Precompiling *DL-Lite*$_\mathcal{A}$ Clash Types

At the end of Section 2.3 we explained how the expansion rules of *DL-Lite*$_\mathcal{A}$ can be used to compute all combinations of concepts, roles and attributes resulting in inconsistencies in an *DL-Lite*$_\mathcal{A}$ ontology. Given a TBox $\mathcal{T}$ we apply this procedure for all clash types storing the results in an efficient index structure. We have to distinguish between clash types that are related to the use of

a) a single concept, role, or attribute (Type 1 and 2),
b) two concepts, two roles, concept and role, or concept and attribute (Type 4),
c) an attribute and a value from a datatype (Type 3),
d) a role or an attribute and the inequality of two instances or values (Type 5 and 6).

For each of these four cases we use a dedicated hash structure (referred to as $\mathbb{H}_a^\perp$ to $\mathbb{H}_d^\perp$), which allows to check if, e.g., the relevant combination of signature elements (concepts, roles, attributes), is contained. We sometimes refer to these data structures in an more general way by omitting the subscript. Once $\mathbb{H}_a^\perp$ to $\mathbb{H}_d^\perp$ have been computed, checking an ABox for consistency breaks down to checking for each single assertion or each pair of assertions if the used signature elements are stored in the respective data structures. We refer to this approach as CQ in the following. This approach has proven itself in practice and was successfully applied in Nolle et al. [14].

3.2 On Demand Reasoning

Our approach, which is a modified extension of CQ, omits the up-front computation of clash queries. Instead, we invoke a reasoner on the fly if we cannot decide the consistency based on clash queries already observed. We refer to that approach as CQ+. CQ+ differs from CQ in three aspects.

First, instead of using the expansion rules, we use a standard OWL 2 reasoner to compute the signature combinations that correspond to instantiations of the clash types. This does not guarantee completeness. There are still possible signature combinations resulting in inconsistencies that are not captured by one of the clash types, e.g., inconsistent combinations of more than two type assertions like $A(a), B(a), C(a)$ with $A \sqcap B \sqsubseteq D$ and $D \sqsubseteq \neg C$. Nevertheless, we will be able to detect inconsistencies that cannot be detected by expanding the stated axioms via the *DL-Lite*$_\mathcal{A}$ expansion rules, because sometimes relevant entailments are based on axioms that are beyond the *DL-Lite*$_\mathcal{A}$ expressivity.

Second, we do not compute inconsistent combinations of signature elements in advance, but on the fly during checking combinations of ABox assertions from the given set of ABoxes. We store every detected clash type instantiation in one of the $\mathbb{H}^\perp$ caches. Note that these caches are empty when we apply the approach to check the consistency of the first ABox. This differs from the CQ approach, where we compute all inconsistent signature combinations in advance. To minimize the calls to the reasoner, we check prior to any reasoning if the currently used combination of signature elements is already stored in one of the $\mathbb{H}^\perp$ caches (or in one of the $\mathbb{H}^\top$ caches, which will be explained in the following paragraph). This will obviously be more effective the more ABoxes are already processed.

Third, we do not only store inconsistent signature combinations, but also consistent combinations. Without this extension we cannot leverage the knowledge about the inconsistencies we detected so far. In the CQ approach we first computed the assertional patterns resulting in inconsistencies, then we started processing the ABoxes checking each assertion or combination of assertions against the pattern stored in $\mathbb{H}^\perp$. If this check is negative, we conclude that the checked combination is consistent. We cannot apply this procedure in the current approach, because $\mathbb{H}^\perp$ will be empty at the beginning and highly incomplete in the initial phase until a significant number of inconsistencies has been observed. For that reason we have to store inconsistent combinations in $\mathbb{H}^\perp$ and consistent combinations in $\mathbb{H}^\top$. For each combination of axioms within an ABox that can found in one of these caches, we decide upon the consistency of this ABox fragment without calling a reasoner.

The algorithm for checking the consistency of an ABox $\mathcal{A}$ against a TBox $\mathcal{T}$ is shown in Algorithm 1. This algorithm is called for each of the ABoxes that need to be checked. The different variants of $\mathbb{H}^\perp$ and $\mathbb{H}^\top$ are referenced via global variables pointing to data structures for which the operations of adding and containment checking run in constant time. We have depicted the algorithm only for Case 4)b). In our actual implementation, we extended the algorithm by a comprehensive set of case distinctions covering the remaining clash types using all four caches.

For the CQ+ approach, every processed ABox increases the probability that a certain combination of signature elements has already been observed to be consistent or

Algorithm 1 CHECKCONSISTENCY($\mathcal{A}, \mathcal{T}$)

```
1:  for all $a \in$ instances($\mathcal{A}$) do
2:      for all $\phi(a) \in$ class-assertions($\mathcal{A}$) do
3:          for all $\psi(a,b) \in$ role-assertions($\mathcal{A}$) do
4:              if $\langle \phi, \exists\psi \rangle \in \mathbb{H}_b^{\perp}$ then
5:                  return false
6:              end if
7:              if $\langle \phi, \exists\psi \rangle \in \mathbb{H}_b^{\top}$ then
8:                  continue // ... with next loop cycle
9:              end if
10:             if $\mathcal{T} \models \phi \sqsubseteq \neg\exists\psi$ then
11:                 $\langle \phi, \exists\psi \rangle \xrightarrow{add} \mathbb{H}_b^{\perp}$
12:                 return false
13:             else
14:                 $\langle \phi, \exists\psi \rangle \xrightarrow{add} \mathbb{H}_b^{\top}$
15:             end if
16:         end for
17:     end for
18: end for
19: return true
```

inconsistent, which means that no reasoning activities are required for that combination. In terms of Algorithm 1, this means that line 10 to 15 will be executed less often the more often the procedure is called. However, this depends both on the size of the TBox and on the distribution of factually used combinations of signature elements, which will be discussed in the following section.

4 Experiments

In Section 4.1 we first describe the datasets used in our experiments and give an overview on the reasoning methods that we evaluate. We present and discuss the most important results of our experiments in Section 4.2.

4.1 Setting

We use two datasets in our experiments[1]. These datasets have also been used in [16] to demonstrate how machine learning can be applied efficiently to mimic a reasoner. We rebuild larger versions of these datasets according to the descriptions in [16]. The first dataset, that we refer to as the DBpedia+ dataset, uses the DBpedia TBox that consists of all mapping-based roles and types in DBpedia [9]. This TBox is extended with the top-level ontology DOLCE-Zero [4]. The reason for this extension is related to the fact that DBpedia contains only few disjointness axioms, while DOLCE-Zero

[1] All the datasets created for this paper are available online at http://web.informatik.uni-mannheim.de/rr2017

Table 1: Characteristics of the DBpedia+ and GoodRelations dataset in terms of number of generated ABoxes, percentage of inconsistent ABoxes, average size of ABoxes; the description logic of the TBox, and number of logical axioms in the TBox.

	ABoxes	Inconsistent	Average Size	DL	Logical Axioms
DBpedia+	100000	24.07%	58.1	$\mathcal{SHIN}(\mathcal{D})$	7436
GoodRelations	5000	28.54%	13.9	$\mathcal{SHI}(\mathcal{D})$	450

introduces disjointness on the top level [15]. We generated 100k ABoxes where each ABox consists of a randomly chosen role assertion and all class assertions related to its subject and object. Thus, all of the ABoxes have a very simple structure and most of the reasoning task is related to checking the asserted types against the domain and range restrictions of the role. However, the explanations for an inconsistency can nevertheless be quite complex. This is illustrated by Example 1, which is an (already slightly simplified) explanation for one of the generated ABoxes. Note also that an ABox usually contains more assertions than shown in Example 1.

The second dataset uses the GoodRelations ontology as TBox. GoodRelations [5] is a vocabulary designed for e-commerce, which is used as RDFa to describe products and offers. We have used a sample of documents from the WebData-Commons 2014 Microdata corpus, and extracted 5k randomly chosen documents (= ABoxes) that make use of the GoodRelations vocabulary. In doing so we encountered syntactic errors and related parsing problems, which required a semi-automated extraction process. For that reason, we were able to extract only 5000 ABoxes. Opposed to the ABoxes from the DBpedia+ dataset, these ABoxes do not share a common structure and are larger compared to the ABoxes from the DBpedia+ dataset. The characteristics of the two datasets are presented in Table 1.

We have not used the schema.org and the YAGO dataset used in [16] because they pose less complex reasoning tasks. According to our inspection of the schema.org dataset, many mistakes are based on roles that have been used as attributes (or vice versa). The remaining inconsistencies seem to be only clashes related to attributes using data values that are incompatible with the explicitly stated data type. Each of the ABoxes in the YAGO dataset is describing a single instance by all of its concept assertions. This means that each ABox is of the form $C_1(a), \ldots, C_2(a)$. Moreover, these assertions are extended in a preprocessing step by adding the transitive closure on the stated class assertions, in order to support the machine learning.

In our experiments we analyze the following reasoning techniques with respect to their performance on the two datasets we introduced above.

CQ refers to the technique that uses the *DL-Lite*$_\mathcal{A}$ expansion rules. We apply this technique to compute inconsistent patterns of assertions in a preprocessing step. Checking an ABox boils down to checking the ABox against these precompiled clash patterns.

CQ+ refers to the method that uses the *DL-Lite*$_\mathcal{A}$ clash patterns. However, instead of using expansion rules in a preprocessing step it uses a complete reasoner to check the relevant entailments on the fly wh3n the corresponding combinations appear in the currently processed ABox. We will also report about experiments, where we turn off the reasoning components after n ABoxes have been processed and the results are than solely based on the cache.

ML refers to the machine learning approach that has been described in [16]. For the feature transformation of the ABoxes, we used all paths up to length 3 for the GoodRelations dataset, while the ABoxes from the DBpedia dataset naturally have a path length of at most 2. All experiments were conducted with RapidMiner Studio.[2] In particular, we report about results using Decision Trees which has turned out to achieve good and stable results.

HermiT is a well-known OWL 2 reasoner [7], which we also used to annotate the training examples for the ML approach.

4.2 Results

The main results of our experiments are depicted in Table 2. The first column is relevant only for CQ+ and ML. For CQ+ it shows the number of ABoxes that have been processed to set up the cache. For ML the same number refers to the number of training examples. The second column shows to the preprocessing time in seconds. In the preprocessing phase the CQ approach computes the expanded clash queries, the ML method trains a classifier which includes also the labeling of the samples with a reasoner (HermiT in our setting), the CQ+ (cache only) method requires processing some ABoxes in order to observe and store clashes (explanations) in the cash, HermiT requires loading the TBox. The third column shows the average time for checking the consistency of an ABox in milliseconds. While we know that all reasoning based methods are sound, i.e., a consistent ABox will never be labeled as inconsistent, an approach based on ML cannot guarantee soundness. For that reason we compare the results in terms of accuracy in the fourth column informing about the fraction of correct answers for each of the methods.

HermiT achieves an accuracy of 100% on both datasets. However, the runtimes are $\approx$25ms and $\approx$100ms for checking a single ABox. This shows that it is problematic to apply standard reasoning techniques if we want to check a very high number of ABoxes, without resorting to parallelization.

The CQ method reaches an accuracy of 98.59% on DBpedia+ and 100% on GoodRelations. An accuracy of 98.59% corresponds, with respect to the DBpedia+ dataset, to 5.44% inconsistent ABoxes that have not been detected to be inconsistent. The runtimes of the CQ approach are about 0.05ms for a DBpedia+ ABox and 0.3ms for a GoodRelations ABox. The differences can be explained by the different size of the ABoxes. Comparing these runtimes to the runtimes of HermiT, the CQ method is about 2000 times faster. The method requires relatively high preprocessing runtimes for the preprocessing step due to the fact that all possible *DL-Lite*$_{\mathcal{A}}$ clashes are computed even though most of them will never be instantiated.

The CQ+ approach is capable of detecting all inconsistencies for both datasets. The improvement for DBpedia+ from 98.59% (CQ) to 100% (CQ+) is caused by the use of

[2] http://www.rapidminer.com

Table 2: Runtime and accuracy of CQ, CQ+, CQ+ turning off reasoning, ML and HermiT for the DBpedia+ and GoodRelations dataset averaged over ten runs for each setting. For settings that depend on previously processed ABoxes ("Training"), we randomly selected the set of these ABoxes in each of the runs.

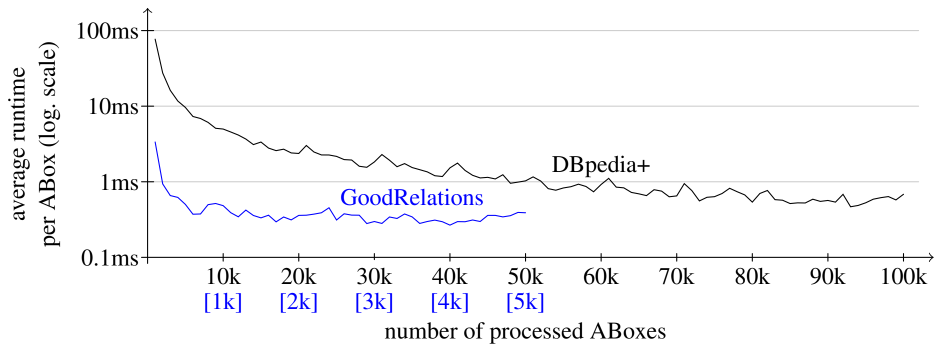
		Training	Runtimes		Accuracy
			Preprocessing (s)	Checking Inc. (ms)	
DBpedia+	CQ	-	746	0.046	98.59%
	CQ+	-	-	77 to $\leq$0.5	100%
	CQ+ (cache only)	1000	77	0.041	98.6%
		10000	172	0.043	99.59%
		50000	253	0.04	99.84%
	ML	1000	98 + 1	0.356	97.62%
		10000	984 + 22	0.383	98.46%
		50000	4919 + 183	0.525	98.52%
	HermiT	-	10	98.38	100%
GoodRelations	CQ	-	20	0.311	100%
	CQ+	-	-	4.5 to $\leq$0.33	100%
	CQ+ (cache only)	50	0	0.318	99.89%
		500	1	0.315	99.92%
		2500	2	0.321	100%
	ML	50	1 + 0	1.483	95.60%
		500	12 + 0	1.589	99.87%
		2500	61 + 1	1.757	99.9%
	HermiT	-	2	24.48	100%

full-fledged reasoning when checking such patterns as $\mathcal{T} \models A \sqsubseteq \neg\exists P$, while the patterns themselves cover all factually existing inconsistencies. The runtimes of the CQ+ approach cannot be presented in terms of an average number, but require presenting the runtime depending on the number of previously checked ABoxes. Remember that with every processed ABox more information is stored in the $\mathbb{H}^{\perp}$ and $\mathbb{H}^{\top}$ caches. To measure the impact of the cache, we apply our algorithm on consecutive blocks of 1000 ABoxes w.r.t DBpedia+ (100 ABoxes w.r.t GoodRelations) measuring the average runtime for a single ABox within such a block.

The resulting runtimes are shown in Figure 1 on a logarithmic scale. We start with a runtime of $\approx$ 80ms for DBpedia. After processing 10k ABoxes the runtime for an ABox is $\approx$ 5ms. After 50k ABoxes have been processed less than 1ms is required. The runtime behavior is similar for GoodRelations. However, significantly less processed ABoxes are required to reduce the initial runtimes. After having processed 1000 ABoxes, the runtimes for CQ and CQ+ are roughly the same. The differences between DBpedia+ and GoodRelations are related to the significantly smaller TBox of GoodRelations. Since there are less concepts, roles and attributes in GoodRelations, there are also less (frequently used) inconsistent and consistent vocabulary combinations.

The CQ method is about 80 times (GoodRelations) to 2000 times (DBpedia+) faster compared to HermiT. This means that by applying the CQ method we are losing completeness, as described above, but gain a significant improvement in runtime. The CQ+

Fig. 1: Runtime of CQ+ for processing a single ABox with respect to the number of already processed ABoxes for DBpedia+ [GoodRelations].



method detects all inconsistencies in both datasets. Its runtimes are at the beginning similar to the runtimes of HermiT. However, after processing a large number of ABoxes, CQ+ is between 80 and 200 times faster compared to HermiT.

One of the most important features of the CQ+ method is the option to turn off reasoning and to rely solely on the cache after a reasonable number of ABoxes have been processed. Table 2 shows the runtimes after turning off reasoning as soon as n ABoxes have been processed. The time required to process the first n ABoxes is counted as preprocessing time in this setting. n is specified in the column entitled ”Training”. For the ML approach we use exactly this number of ABoxes as training examples. The CQ+ runtimes, after turning off the reasoning component, are approximately the same runtimes that we measured for the CQ approach. However, the accuracy is surprisingly high after processing a relatively small number of ABoxes. After processing 1k ABoxes of DBpedia+, the accuracy is similar to the accuracy of the CQ method; after 10k ABoxes we are already reaching an accuracy of 99.59% which is increased to 99.84% when processing 50k ABoxes. For GoodRelations we achieve an accuracy of 100% after having processed 2500 ABoxes. This shows that the CQ+ method is highly flexible and can be configured for the needs of a given application scenario (runtime vs. accuracy).

We have also applied the ML approach of Paulheim and Stuckenschmidt [16]. The runtimes are slightly worse compared to the runtimes of the CQ method. Note that most of the runtimes are related to generating the feature representation of an ABox, while the classification itself is extremely fast. The high runtimes for preprocessing are caused by the need to annotate the training examples with the help of HermiT, while the runtimes for learning the classifier are of little significance. The accuracy of the approach is rather high. However, the ML results that are based on learning from n examples are worse than comparable results of the CQ+ approach relying solely on the cache after n ABoxes have been processed with an active reasoning component. This becomes more evident when we present the results in terms of the error rate $(1 - accuracy)$. The error rate on DBpedia+ with $n = 10000$ is 1.54% for ML and 0.41% for CQ+, the error rate on GoodRelations with $n = 2500$ is 0.13% for ML and 0.08% for CQ+. This

Table 3: Characteristics related to memory consumption for CQ, CQ+ and ML.

	DBpedia		GoodRelations	
# processed/training examples	10k	100k	500	5000
CQ ($\mathbb{H}_b^\top$ / $\mathbb{H}_b^\perp$)	- / 8810234		- / 11184	
CQ+ ($\mathbb{H}_b^\top$ / $\mathbb{H}_b^\perp$)	17057 / 480	27580 / 1460	804 / 7	1722 / 12
ML (number of features)	4211.6	8261	1781.4	2427

illustrates the theoretical considerations that we presented in Section 2.4. Moreover, the accuracy of the learned classifier is in none of the settings higher than the well known CQ method.

In Table 3 we present numbers relevant to the memory usage. For CQ and CQ+ we show the number of stored combinations in the $\mathbb{H}_b^\top$ (consistent usage) and $\mathbb{H}_b^\perp$ (inconsistent usage). Note that the $\mathbb{H}_b$ caches are the only ones that can increase up to the square of the concepts, roles and attributes defined in the TBox, while the other caches will only grow linearly. For ML we show the dimension of the feature vector that describes a single ABox. First, we compare the clash patterns stored by CQ and CQ+. While there are more than 8 million instantiations of clashes, there occur only 1460 of these clashes in a set of 100k ABoxes. This is less then 0.02%. For GoodRelations we measured only 12 different inconsistency patterns.[3] The fact that only very specific errors occur in the dataset is also a reason why learning works in the given setting. The fact that DBpedia results in more than 8 million inconsistent combinations shows that the CQ approach is only applicable to larger TBoxes if extensive memory resources are available. While the number of consistent combinations clearly dominates the inconsistent combinations for CQ+, the overall cache size is still acceptable and seems to grow linearly with respect to the size of the TBox. The number of features in the ML approach is less for DBpedia+ and more for GoodRelations compared to the sum of the entries in the CQ+ caches. Overall, the numbers are in the same order of magnitude. The differences are mainly based on the fact that the maximal path length in the DBpedia+ ABoxes is two, while the GoodRelations dataset has ABoxes that correspond to graphs with longer paths.

5 Conclusion and Future Work

In this work, we have studied different methods to solve the problem of efficiently checking the consistency of a large set of ABoxes against a given TBox. Our results indicate that the approach proposed in [11], which is complete for *DL-Lite*$_\mathcal{A}$, can also be applied successfully to scenarios where the given TBox is more expressive. While such an approach, referred to as CQ, is incomplete in theory, our experiments indicted that only few inconsistencies remain undetected. Moreover, the CQ approach clearly

[3] Note that the $\mathbb{H}_b$ caches do not contain errors related to wrong datatypes. However, these errors are less important from a reasoning perspective since all of them have been detected by comparing the stated datatype against the type of the given value.

outperforms an approach based on machine learning, which has been proposed more recently [16].

We extended the CQ resulting in an approach referred to as CQ+. This approach is based on the use of a complete reasoner to check on the fly the entailment that results in an instantiation of a clash pattern. Spotted inconsistent and consistent combinations of vocabulary usage are stored in a cache, which is used prior to any calls to the reasoner. This extension resulted in an accuracy of 100% for both datasets used in our experiments. Moreover, we could show that the runtimes decrease with every processed ABox resulting in a highly efficient procedure for checking consistency.

Even though we measured in our experiments an accuracy of 100%, we are aware that this is only an empirical observation related to the datasets we used in our experiments. Even within the simple structure of the DBpedia ABoxes, there is no guarantee that the CQ+ method finds all inconsistencies. One can easily define sets of axioms that would result, in combination with an ABox that has the same structure as the ABoxes of the DBpedia+ dataset, in an inconsistency. An example are the following axioms and assertions.

$$\exists P.B \sqsubseteq A, A \sqsubseteq \neg C, P(a,b), C(a), B(b)$$

Both CQ and CQ+ are not capable of detecting this inconsistency, due to the fact that the involved assertions do not instantiate one of the clash types. A machine learning based approach is in principle capable of learning a classifier that will work for such cases. This will happen if instantiations of $P(x,y)$, $C(x)$, $B(y)$ appear sufficiently often within the training examples, while proper subsets of these instantiations, that are marked as consistent, will also appear sufficiently often within the training examples. Other inconsistencies that are beyond the scope of CQ+ involve, for example, role transitivity and role irreflexivity.

In general, an approach that is based on learning can pay-off only for datasets where two conditions hold:

- Some inconsistencies in the dataset cannot be detected with incomplete but efficient reasoning techniques.
- These inconsistencies are instantiations of the same pattern that appears rather frequently in the dataset.

In such a setting it might make sense to use (additionally) an approach that leverages machine learning techniques to learn patterns that are not covered by CQ, CQ+ or any alternative inference method. This requires the development of an appropriate feature representation without introducing a huge feature space that cannot be handled efficiently. However, unless such a method based on machine learning is available, it seems to be the best choice to rely on the CQ method, the CQ+ method, or another elementary inference method.

As a first step in our future work, we plan to use the reasoner Konclude [18] in our experiments.

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. Flouris, G., Huang, Z., Pan, J.Z., Plexousakis, D., Wache, H.: Inconsistencies, negations and changes in ontologies. Proceedings of the National Conference on Artificial Intelligence 21(2), 1295 (2006)
4. Gangemi, A., Mika, P.: Understanding the semantic web through descriptions and situations. In: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems". pp. 689–706. Springer (2003)
5. Hepp, M.: Goodrelations: An ontology for describing products and services offers on the web. In: International Conference on Knowledge Engineering and Knowledge Management. pp. 329–346. Springer (2008)
6. Horridge, M., Parsia, B., Sattler, U.: Explaining inconsistencies in owl ontologies. In: International Conference on Scalable Uncertainty Management. pp. 124–137. Springer (2009)
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)
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. Lehmann, J., Isele, R., Jakob, M., Jentzsch, A., Kontokostas, D., Mendes, P.N., Hellmann, S., Morsey, M., Van Kleef, P., Auer, S., et al.: DBpedia–a large-scale, multilingual knowledge base extracted from wikipedia. Semantic Web 6(2), 167–195 (2015)
10. 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)
11. 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)
12. Lösch, U., Bloehdorn, S., Rettinger, A.: Graph kernels for rdf data. In: Extended Semantic Web Conference. pp. 134–148. Springer (2012)
13. Nickel, M., Murphy, K., Tresp, V., Gabrilovich, E.: A review of relational machine learning for knowledge graphs. Proceedings of the IEEE 104(1), 11–33 (2016), http://dx.doi.org/10.1109/JPROC.2015.2483592
14. Nolle, A., Meilicke, C., Chekol, M., Nemirovski, G., Stuckenschmidt, H.: Schema-based debugging of federated data sources. In: Proceedings of the 22nd European Conference on Artificial Intelligence (ECAI2016). IOS Press (2016)
15. Paulheim, H., Gangemi, A.: Serving dbpedia with dolce – more than just adding a cherry on top. In: International Semantic Web Conference. pp. 180–196. Springer (2015)
16. Paulheim, H., Stuckenschmidt, H.: Fast approximate a-box consistency checking using machine learning. In: Extended Semantic Web Conference. pp. 135–150. Springer (2016)
17. 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)
18. Steigmiller, A., Liebig, T., Glimm, B.: Konclude: System description. J. Web Sem. 27, 78–85 (2014), http://dx.doi.org/10.1016/j.websem.2014.06.003