Fast Computation of Explanations for Inconsistency in Large-Scale Knowledge Graphs

Trung-Kien Tran  
Bosch Center for Artificial Intelligence  
Renningen, Germany  
trungkien.tran@de.bosch.com

Mohamed H. Gad-Elrab  
Bosch Center for Artificial Intelligence  
Renningen, Germany  
max-planck institute for informatics  
Saarbrücken, Germany  
gadelrab@mpi-inf.mpg.de

Daria Stepanova  
Bosch Center for Artificial Intelligence  
Renningen, Germany  
daria.stepanova@de.bosch.com

Evgeny Kharlamov  
Bosch Center for Artificial Intelligence  
Renningen, Germany  
evgeny.kharlamov@de.bosch.com

Jannik Strötgen  
Bosch Center for Artificial Intelligence  
Renningen, Germany  
jannik.stroetgen@de.bosch.com

ABSTRACT

Knowledge graphs (KGs) are essential resources for many applications including Web search and question answering. As KGs are often automatically constructed, they may contain incorrect facts. Detecting them is a crucial, yet extremely expensive task. Prominent solutions detect and explain inconsistency in KGs with respect to accompanying ontologies that describe the KG domain of interest. Compared to machine learning methods they are more reliable and human-interpretable but scale poorly on large KGs. In this paper, we present a novel approach to dramatically speed up the process of detecting and explaining inconsistency in large KGs by exploiting KG abstractions that capture prominent data patterns. Though much smaller, KG abstractions preserve inconsistency and their explanations. Our experiments with large KGs (e.g., DBpedia and Yago) demonstrate the feasibility of our approach and show that it significantly outperforms the popular baseline.

ACM Reference Format:

1 INTRODUCTION

Motivation. A Knowledge Graph (KG) describes facts about entities by labeling entities with classes and interconnecting them via binary relations. For instance, the top of Figure 1 depicts a KG with facts about companies and locations, e.g., Nokia is a PopularName, and it is a customer of IBM. Prominent examples of large-scale KGs include DBpedia [19], NELL [23], Yago [31], Freebase [3], and Wikidata [34] – all contain millions of facts. KGs power a wide variety of important applications including Web search [5] and question answering [6]. While data quality of a KG is crucial for such applications, unfortunately large-scale KGs are often inaccurate. Indeed, they are constructed using error-prone methods such as information extraction (e.g., Yago), crowd-sourcing (e.g., Wikidata) or curated using embedding-based approaches (see [36] for an overview). These methods inevitably introduce erroneous information, e.g., incorrectly disambiguated entities or faulty relations.

A common task in such settings is that of detecting incorrect facts in KGs. Although many works on this topic have focused on machine-learning techniques (e.g., [15, 27]), methods that rely on symbolic reasoning (e.g., [25, 32]) have shown the benefit for this task due to their accuracy and human-interpretable nature [26]. One prominent such method for detecting incorrect facts in KGs is to compute inconsistency explanations of the KG with respect to a manually crafted ontology [14], which is a set of axioms that capture the relevant domain of interest and constraints enforcing that some statements must not occur together. An example ontology presented at the bottom of Figure 1 states that the range of hasCustomer relation is a Company, the domain of isCityOf relation is a City, and the classes City, Company and Country are disjoint. Such ontologies are usually carefully created by domain experts, and the errors in the KGs are typically characterized by whether there exist triples in the KG that contradict with the accompanied ontology.

It is easy to verify that our sample KG contradicts the ontology, as Toyota and Nokia belong to disjoint classes Company and City. A minimal set of axioms causing this is an inconsistency explanation.

Detecting and explaining inconsistency in KGs enhanced with ontologies are fundamental and well-studied knowledge management tasks [13, 26]. It is well-known that finding all and even some explanations is computationally demanding and thus does not scale
for large real-world KGs [28]. This holds even for light-weight ontology languages that admit scalable processing since the number of explanations can be huge: in DBpedia there are at least 3,500,000 such explanations [28].

**General Idea.** We propose a novel method to compute inconsistency explanations for large-scale KGs. The key idea of our method is to identify an ontology language and construct a compressed representation, called abstraction (abstract KG), of the original KG such that for any ontology written in that language, the following requirements are satisfied:

[R1] **preserve KG’s consistency:** the original KG is consistent with respect to the ontology iff its abstraction is so,

[R2] **preserve KG’s explanations:** inconsistency explanations for the abstraction can be used to obtain exactly all inconsistency explanations for the original KG, and

[R3] **be smaller than KG:** the size of the abstraction is much smaller than the size of the original KG.

Then, instead of checking consistency and computing the explanations directly on the original KG (w.r.t. an ontology), we perform such operations on its abstraction. The requirements R1–R2 guarantee the correctness of our method while R3 ensures its scalability.

**Contributions.** We show the existence of such abstractions for a fairly expressive fragment of the Web Ontology Language (OWL 2). More precisely, our contributions are as follows:

- Following the requirements R1–R3 we propose and formalize an abstraction-based framework for computing explanations for inconsistency of ontology-enhanced KGs.
- We identify a fragment of OWL 2 for which inconsistency explanations can be computed by partitioning the input KG into small independent modules.
- For the identified OWL 2 fragment, we develop an efficient algorithm to compute all explanations for inconsistent knowledge graphs based on data abstraction.
- We evaluate our approach on NPD [30], Yago [31], and DBpedia [19] KGs with very promising results: we manage to compute thousands of inconsistency explanations and their patterns, for which computing directly on the original KGs is infeasible.

## 2 PRELIMINARIES

**Knowledge Graphs and Ontologies.** We assume countable pairwise disjoint sets $N_C$, $N_P$ and $N_I$ of class names, e.g., Company, property names, e.g., hasCustomer, and individuals, e.g., Toyota. A knowledge graph (KG) $G$ is a finite set of unary and binary facts, also called assertions, of the form $C(a)$ and $R(a,b)$, where $C \in N_C$, $R \in N_P$ and $a, b \in N_I$, e.g., Popularity(Nokia) or hasCustomer(McAfee,Nokia).

We use $\text{ind}(G)$ to denote the sets of individuals occurring in $G$. From class and property names one can (recursively) define complex classes $C$ and properties $P$ following the OWL 2 specification. In this work, we focus on the following types of classes and properties:

$$ P ::= R \mid \text{ObjectInverseOf}(P) $$

$$ C ::= \text{owl:Thing} \mid \text{owl:Nothing} \mid A \mid \text{ObjectComplementOf}(C) \mid \text{ObjectIntersectionOf}(C_1, C_2) \mid \text{ObjectUnionOf}(C_1, C_2) \mid \text{ObjectSomeValuesFrom}(P, \text{owl:Thing}), $$

where $A \in N_C$, $R \in N_P$, $C_1, C_2$ are classes and $P$ is a property. Classes and properties can be used to define axioms that formally describe the domain of interest, and the set of such axioms is called an ontology. Table 1 describes the axioms considered in this paper. Note that these axioms can express other common OWL 2 axioms such as: InverseObjectProperties, ObjectPropertyDomain, ObjectPropertyRange, EquivalentClasses, and DisjointClasses.

**Inconsistency and Explanations.** The semantics of knowledge graphs and ontologies is defined using the OWL 2 direct model-theoretic semantics via interpretations [24]. An interpretation $I = (\Delta^I, \cdot)$ consists of a non-empty set $\Delta^I$, the domain of $I$, and an interpretation function $\cdot$, that assigns to each $A \in N_C$ a subset $\Delta^I \subseteq \Delta^I$, to each $R \in N_P$ a binary relation $R^I \subseteq \Delta^I \times \Delta^I$, and to each $a \in N_I$ an element $a^I \in \Delta^I$. This assignment is extended to (complex) classes and properties as shown in Table 1.

An interpretation $I$ satisfies an axiom $\alpha$ (written $I \models \alpha$) if the corresponding condition in Table 1 holds. Given a KG $G$ and an ontology $O$, $I$ is a model of $G \cup O$ (written $I \models G \cup O$) if $I \models \alpha$ for all axioms $\alpha \in G \cup O$. We say that $G \cup O$ entails an axiom $\alpha$ (written $G \cup O \models \alpha$), if every model of $G \cup O$ satisfies $\alpha$. A knowledge graph $G$ is inconsistent w.r.t. an ontology $O$ if no model for $G \cup O$ exists. In this case, we also say $G \cup O$ is inconsistent. Intuitively, $G \cup O$ is inconsistent when some facts of $G$ contradict some axioms of $O$.

An explanation for inconsistency of $G \cup O$ [13], denoted by $E = E_G \cup E_O$ with $E_G \subseteq G$ and $E_O \subseteq O$, is a smallest subset-inclusion inconsistent subset of $G \cup O$. E.g., the KG part in Figure 1 stating that McAfee hasCustomer Toyota and Toyota isCityOf Japan together with the three example ontology axioms is an explanation. In general, inconsistency of $G \cup O$ may have multiple explanations.

**Homomorphisms.** We now define homomorphisms between KGs to establish the theoretical foundation for our approach.

**Definition 2.1.** Let $G_1$ and $G_2$ be knowledge graphs and $h$: $\text{ind}(G_1) \rightarrow \text{ind}(G_2)$ a mapping from the individuals occurring in $G_1$ to individuals occurring in $G_2$. We extend $h$ to axioms in a straightforward way: $h(C(a)) = C(h(a))$, $h(R(a,b)) = R(h(a), h(b))$. We define $h(O) = O$ for every ontology $O$ and define $h(G_1) = \{h(\alpha) | \alpha \in G_1\}$. We say that $h$ is a homomorphism (from $G_1$ to $G_2$) if $h(G_1) \subseteq G_2$.

The following well-known lemma states that entailment is preserved under homomorphisms.

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>$R$</td>
<td>$R^I \subseteq \Delta^I \times \Delta^I$</td>
</tr>
<tr>
<td>ObjectInverseOf($P$)</td>
<td>${ (e, d)</td>
</tr>
<tr>
<td>$A$</td>
<td>$A^I \subseteq \Delta^I$</td>
</tr>
<tr>
<td>owl:Thing</td>
<td>$\Delta^I$</td>
</tr>
<tr>
<td>owl:Nothing</td>
<td>$\emptyset$</td>
</tr>
<tr>
<td>ObjectComplementOf($C$)</td>
<td>$\Delta^I \setminus C^I$</td>
</tr>
<tr>
<td>ObjectIntersectionOf($C_1, C_2$)</td>
<td>$C_1^I \cap C_2^I$</td>
</tr>
<tr>
<td>ObjectUnionOf($C_1, C_2$)</td>
<td>$C_1^I \cup C_2^I$</td>
</tr>
<tr>
<td>ObjectSomeValuesFrom($P, \text{owl:Thing}$)</td>
<td>${ d \mid \exists e \in \Delta^I : (d, e) \in P^I }$</td>
</tr>
<tr>
<td>SubClassOf($C, D$)</td>
<td>$C^I \subseteq D^I$</td>
</tr>
<tr>
<td>SubObjectPropertyOf($P, S$)</td>
<td>$P^I \subseteq S^I$</td>
</tr>
<tr>
<td>TransitiveObjectProperty($P$)</td>
<td>$P^I \circ P^I \subseteq P^I$</td>
</tr>
<tr>
<td>ClassAssertion($C, a$)</td>
<td>$a^I \in C^I$</td>
</tr>
<tr>
<td>ObjectPropertyAssertion($R, a, b$)</td>
<td>$\langle a^I, b^I \rangle \in R^I$</td>
</tr>
</tbody>
</table>

Table 1: Syntax and semantics of the ontology language considered in this paper where $A, R$ are a class name and property name, respectively; $C$ and $D$ are class expressions, $P, S$ are property expressions, and $a, b$ are individuals.
Given a set $a$ tuple corresponding module for that individual. The local type of an individual in a knowledge graph is identical $a$ and there exists no $t \in \{a\}$ such that it can be homomorphically mapped $a$ module (Sec. 3.1). In Step 2, we detect commonalities across modules, which is occurring in the property exploited in Step 3 of our approach. To characterize the definition 3.5.

3.3.1 Modules and Inconsistency Locality Property

In KGS, entities are highly connected, and therefore, ontology axioms could trigger multi-hop reasoning steps, e.g., the transitivity axiom TransitiveObjectProperty(foaf), where foaf stands for friend of a friend, combined with two facts foaf(John, Mary) and foaf(Mary, John) in a KG infers a new fact foaf(John, Mary). However, under the considered ontology language, we discovered that the KG inconsistency has a locality property, i.e., the problem of checking inconsistency for a KG (w.r.t. an ontology $O$) can be reduced to checking inconsistency for separated KG modules (w.r.t. $O$).

Definition 3.1 (Modules). Given a KG $G$ and an individual $a \in \text{ind}(G)$, the module of $a$ w.r.t. $G$ is defined as $M(a, G) = \{a | a \in G$ and $a$ occurs in $a\}$. We denote the set of all modules for individuals occurring in $G$ as $M_G = \{M(a, G) | a \in \text{ind}(G)\}$.

Intuitively, the module for an individual $a$ in a knowledge graph is the set of all facts containing $a$. In our example in Figure 1, the module of Nokia is defined as $M(\text{Nokia}, G) = \{\text{PopularName(Nokia), hasCustomer(McAffee, Nokia), isCityOf(Nokia, Japan)}\}$.

We demonstrated Lemma 3.2. Let $G$ be a KG and $O$ an ontology. Then $G \cup O$ is consistent if $M(a, G) \cup O$ is consistent for every $a \in \text{ind}(G)$.

Before proving Lemma 3.2 we show the following proposition.

**Proposition 3.3.** Let $G$ be a knowledge graph such that $(R(a, b'), R(d, b)) \subseteq G$ for some property $R$ and individuals $a, b, a', b'$. Furthermore, let $G' = G \cup (R(a, b))$. Then, for every ontology $O$, we have $G \cup O$ is consistent implies $G' \cup O$ is consistent.

**Proof of Proposition 3.3.** We assume $G \cup O$ is consistent and show that $G' \cup O$ is consistent. The idea is to construct a model of $G' \cup O$ based on a model of $G \cup O$. Let $I$ be a model of $G \cup O$. We construct an interpretation $J$ as follows: $A^J = A^I; a^J = a^I$ and $A^J = A^I$ for every individual $a$ and class name $A$, respectively; and for every property name $P$, we define: $P^J = P^I \cup \{(a^I, b^I) \in O \models \text{SubObjectPropertyOf}(R, P)\}$.

We show $J$ entails all axioms in $G \cup O$ except for the property transitivity axioms. Since $I \models G \cup O$ and the interpretation of class names and individuals remains the same in $J$, we have $J$ entails each class assertion in $G'$. Furthermore, from the definition of $J$, the interpretations of properties are expanded such that $(a^I, b^I) \in R^J$. Therefore, $J$ entails each property assertion $R(a, b)$ in $G'$. It remains to show that $J \models O$. We prove by induction that, for every class expression $D$, we have $D^J = D^I$.

- Cases $D = \text{owl:Thing}, D = \text{owl:Nothing}$, and $D = A$, where $A \in NC$ follow directly from the definition of $J$.
- Case $D = \text{ObjectSomeValuesFrom}(R, \text{owl:Thing})$. We consider the case, where $R$ is a property name: the case of inverse property is similar. Let $d$ be an arbitrary element in $D^J$. According to the semantics of existential axioms represented in Table 1, we have $d \in \text{ObjectSomeValuesFrom}(R)$, if $d$ holds for $d$ is a model of $G \cup O$, we obtain $(a^I, b^I) \in P^J$. Therefore, $d = a^I \in D^J$. Since $d$ is arbitrary, we have $D^J \subseteq D$. Furthermore, since the interpretations of properties are expanded, we get $D^J \subseteq D^I$. Hence, $D^J = D^I$.
- Cases $D = \text{ObjectComplementOf}(D'), D = \text{ObjectIntersectionOf}(D_1, D_2)$, and $D = \text{ObjectUnionOf}(D_1, D_2)$ follow from their semantics and the induction hypothesis.

For every subclass axiom $\text{SubClassOf}(C_1, C_2) \in O$, we have $C_1^J \subseteq C_2^J$. Therefore, $J \models \text{SubClassOf}(C_1, C_2)$. The interpretation of properties are expanded according to the sub property axioms, $J$ entails every subproperty axiom in $O$.

In the last step, $J$ is extended to obtain $J'$ by expanding the interpretations of properties such that $J'$ entails every transitivity axiom in $O$. One could repeat the previous steps to show that $J'$ still entails other axioms in $O$. Therefore, $J' \models G' \cup O$, which implies that $G' \cup O$ is consistent.

We are now ready to show the proof for Lemma 3.2.

**Proof of Lemma 3.2.** The “only-if direction” is trivial since each module of $G$ is a subset of $G$, and therefore, $G \cup O$ is consistent implies all modules are consistent. We prove the remaining direction by assuming that $M(a, G) \cup O$ is consistent for all $a \in \text{ind}(G)$; we
then show that $\mathcal{G} \cup O$ is also consistent. This is done via the following steps. First, for each module, e.g., $M(a, \mathcal{G})$ of $\mathcal{G}$, we rename all individuals in it except for a s.t. individuals occurring in each module are pairwise disjoint. It allows us to use the disjoint union of the models to construct an interpretation which is close to a model of $\mathcal{G} \cup O$. For each module $M(a, \mathcal{G})$, we denote the resulting renamed module as $M'(a, \mathcal{G})$, and since $M(a, \mathcal{G}) \cup O$ is consistent, we have $M'(a, \mathcal{G}) \cup O$ is consistent (by Lemma 2.2). Let $I_\mathcal{G}$ be a model of $M'(a, \mathcal{G}) \cup O$ for each individual $a \in \text{ind}(\mathcal{G})$. We define an interpretation $I$ as the disjoint union of all $I_\mathcal{G}$ for all individuals $a$ occurring in $\mathcal{G}$. Then, $I = \{M'(a, \mathcal{G}) \mid a \in \text{ind}(\mathcal{G})\} \cup O$, which implies that $G_1 \cup O$, where $G_1 = \bigcup_{a \in \text{ind}(\mathcal{G})} M'(a, \mathcal{G})$, is consistent.

Second, we extend $G_1$ by adding all property assertions in $E$ to obtain $G_2$ defined as $G_2 = G_1 \cup \{\text{ObjectPropertyAssertion}(R,a,b) \mid \text{ObjectPropertyAssertion}(R,a,b) \in G_1\}$. By Proposition 3.3, we have $G_2 \cup O$ is consistent. Finally, by monotonicity, we obtain $\mathcal{G} \cup O$ is consistent because $\mathcal{G} \subseteq G_2$ and $G_2 \cup O$ is consistent. □

Using Lemma 3.2, we can show that all explanations for inconsistency of a KG can be obtained by computing explanations for its modules and thus in Step 1 we construct modules.

**Theorem 3.4.** Let $\mathcal{G}$ be a KG and $O$ an ontology s.t. $\mathcal{G} \cup O$ is inconsistent. Then, $E$ is an explanation for inconsistency of $\mathcal{G} \cup O$ iff there exists $a \in \text{ind}(\mathcal{G})$ s.t. $M(a, \mathcal{G}) \cup O$ is inconsistent and $E$ is an explanation for the inconsistency of $M(a, \mathcal{G}) \cup O$.

**Proof of Theorem 3.4.** The “if” direction of Theorem 3.4 holds as, by monotonicity, all explanations for inconsistency of the modules of $\mathcal{G}$ are explanations for the inconsistency of $\mathcal{G}$. We now show the “only-if” direction. Let $E = E_\mathcal{G} \cup E_O$, where $E_\mathcal{G} \subseteq \mathcal{G}$ and $E_O \subseteq O$, is an inconsistency explanation for $\mathcal{G} \cup O$. Since $E$ is inconsistent, by Step 3, an individual $a$ occurring in $G_\mathcal{G}$ exists, such that $M(a, E_\mathcal{G}) \cup O \subseteq \mathcal{G}$ is inconsistent. Let $E'$ be an explanation for inconsistency of $M(a, E_\mathcal{G}) \cup O \subseteq \mathcal{G}$, which is also an explanation for $\mathcal{G} \cup O$. Since $E' \subseteq M(a, \mathcal{G}) \cup E_O \subseteq \mathcal{G}$ and $E$ is minimal, we have $E = E'$, which is what is required to be proved. □

Using Theorem 3.4, one can already compute inconsistency explanations for a KG based on its modules, even without further steps. We show in Section 4 that this gives a significant speed-up over the baseline. More importantly, we observe that many modules have similar structures and computing inconsistency explanations for multiple modules can be performed on just one representative module. This motivates the further steps for KG abstraction.

### 3.2 Graph Abstraction

Let $N_\mathcal{G}$ be a countable set of abstract individuals that is disjoint with $N_\mathcal{C}$, $N_\mathcal{P}$, and $N_\mathcal{O}$. An abstract KG is a KG consisting of facts over abstract individuals, i.e., a finite set of facts of the form $C(u)$, $R(u, v)$, where $u, v \in N_\mathcal{G}, C \subseteq N_\mathcal{C}, R \subseteq N_\mathcal{P}$. Intuitively, each abstract individual is a representative for several ones in the original KG.

Given an ontology $O$ and a KG $\mathcal{G}$, an abstract graph $G'$ is inconsistency preserving (for $\mathcal{G}$) when $G'$ is consistent (w.r.t. $O$) iff $\mathcal{G}$ is consistent. An abstract graph $G'$ is explanation preserving (for $\mathcal{G}$) if for every inconsistency explanation $E$ of $\mathcal{G}$, there exists an inconsistency explanation $E'$ of $G'$ such that $E'$ can be homomorphically mapped to $E$. An inconsistency and explanation preserving abstract graph $G'$ (for $\mathcal{G}$) is called an abstraction of $\mathcal{G}$. Clearly, an abstraction of any KG satisfies the requirements R1–R2, but it does not always guarantee to satisfy R3, which is important for handling large KGs. In the following, we propose a method for constructing KG abstractions that are significantly smaller than the original real-world KGs, i.e., R3 holds for them in practice (see Section 4).

The abstraction of a KG is computed by identifying modules with similar structures and then representing each group of similar modules by just a single representative one. To characterize the structure of modules, we rely on the local type of individuals [9, 11]. Intuitively, a local type of an individual represents a set of classes and the incoming and outgoing properties occurring in the corresponding module for that individual.

**Definition 3.5 (Local Types).** Let $\mathcal{G}$ be a KG and $a \in \text{ind}(\mathcal{G})$, then, the local type of a w.r.t. $G$, written as $\tau(a, \mathcal{G})$ or $\tau(a)$ when $\mathcal{G}$ is clear from the context, is defined as a tuple $\tau(a) = \langle \tau_1(a), \tau_2(a), \tau_3(a) \rangle$, where $\tau_1(a) = \{R \mid R(c, a) \in G\}$, $\tau_2(a) = \{A \mid A(a) \in G\}$, and $\tau_3(a) = \{S \mid S(a, b) \in G\}$. The local type $\tau$ is defined as $\tau_i \subseteq \tau_i'$ for each $\tau_i, \tau_i' \subseteq \tau$, and $\tau$ is inconsistent if $\tau_i \notin \tau_i' \cup \tau_i' \cup \tau_i$ for some $\mathcal{G}$. A given set $\mathcal{S}$ of local types is a local type $\tau$ is called maximal (in $\mathcal{S}$) if $\tau \in \mathcal{S}$ and no $\tau'$ s.t. $\tau \notin \mathcal{S}$ exists.

Each local type consists of sets of classes and properties. We now define a (star shape) abstraction for each local type by instantiating these classes and properties with abstract individuals in $N_\mathcal{G}$.

**Definition 3.6 (Star Shape Abstractions).** Let $\mathcal{G}$ be a KG and $a \in \text{ind}(\mathcal{G})$, then, the star shape abstraction for $\tau$ is defined as $\mathcal{G}(\tau) = \{A(v) \mid A \in \mathcal{S}\} \cup \{R(u, v) \mid R \in \mathcal{S}\} \cup \{S(u, w) \mid S \in \mathcal{S}\}$, where $u, v, w$. $\tau$ are abstract individuals for each $\tau, R, S$, and $\mathcal{V}$.

For simplicity, we will omit the subscripts and superscripts of individuals in star shape abstractions. These are only needed to formally make those individuals distinguishable.

**Example 3.7.** Toyota and Nokia in Figure 1 have the same local type defined as $\tau = \langle \{\text{PopularName}\}, \{\text{hasCustomer}\}, \{\text{isCityOf}\} \rangle$. For $\tau$, $\mathcal{S}(\tau) = \{\text{PopularName}(u), \text{hasCustomer}(u, v), \text{isCityOf}(v, w)\}$ is the star shape abstraction.
We now define the \( \tau \) can be homomorphically mapped to any realization of \( \tau \) (in Algorithm 1) is an inconsistency explanation of \( G \cup O \). A realization for the abstractions and three corresponding explanations (i.e., the realizations of \( \tau(v_\ell, E_G) \), for the original KG in Figure 1. The former is regarded as explanation pattern for the latter.

### 3.3 Inconsistency Explanation Reconstruction

We now define the realization of a local type that allows us to reconstruct explanations for the original KG from those for abstractions.

**Definition 3.8 (Realization of a local type).** Let \( G \) be a KG and \( \tau' \) a local type. A realization of \( \tau' \) for an individual \( a \in \text{ind}(G) \) is an inclusion-smallest subset \( \text{real}_{\tau'} \) of \( G \) s.t. \( \tau(a, \text{real}_{\tau'}) = \tau' \). The realization of \( \tau' \) in \( G \) is the set of all realizations of \( \tau' \) for each individual occurring in \( G \).

In Figure 3, we present the KG-part \( E_G \) of an inconsistency explanation for the abstractions and three corresponding explanations (i.e., the realizations of \( \tau(v_\ell, E_G) \)), for the original KG in Figure 1. The former is regarded as explanation pattern for the latter.

### 3.4 Bringing all Together

Finally, we proceed with Algorithm 1 that computes all explanations for KG \( G \) w.r.t. an ontology \( O \) using abstraction. Since it iterates over local types, whose number is bounded by \( |\text{ind}(G)| \), and the number of explanations for abstractions as well as the number of realizations for types are bounded by the signature of \( G \) and \( O \), Algorithm 1 terminates. Figure 3 illustrates all steps of the algorithm for the example in Figure 1. The following theorem shows its correctness.

**Theorem 3.9.** Given a KG \( G \) and an ontology \( O \) as inputs for Alg. 1, \( \cup_{a \in \text{ind}(G)} \text{abs}(\tau(a)) \) is an abstraction of \( G \) and the set \( \text{allExpls} \) consists of all inconsistency explanations for \( G \) w.r.t. \( O \).

**Proof of Theorem 3.9.** The abstraction \( \text{abs}(\tau) \) for every type \( \tau \) can be homomorphically mapped to any realization of \( \tau \) in \( G \). Therefore, by Lemma 2.2, if \( \tau' \) is the local type of some individual \( v_\ell \) in an inconsistency explanation of \( \text{abs}(\tau) \cup O \), then the realizations of \( \tau' \) (together with ontology axioms) in \( G \) are inconsistency explanations of \( G \cup O \). This shows that every element of \( \text{allExpls} \) (in Algorithm 1) is an inconsistency explanation of \( G \cup O \).

To prove that \( \text{allExpls} \) contains all explanations of \( G \cup O \), we rely on the claim that \( G \cup O \) is consistent if \( \text{abs}(\tau(a)) \cup O \) is consistent for every individual \( a \) in \( G \), which can be proved based on Proposition 3.3 as done in the proof of Lemma 3.2. Then, similar to the proof of Theorem 3.4, we show that every explanation for inconsistency of \( G \cup O \) can be obtained via some inconsistency explanation of \( \text{abs}(\tau) \cup O \) for some maximal local type \( \tau \).

### 4 EXPERIMENTS

We have implemented our method for computing inconsistency explanations in a system prototype and evaluated it on three real world KGs equipped with their ontologies: the Norwegian Petroleum Directorate (NPD) [30], Yago [31], and DBpedia [19]. Since the original version of the NPD dataset is consistent, for the purpose of our experiments we introduced inconsistency by adding 2% of random property and class assertions. We used the ontologies provided with NPD and DBpedia KGs. For Yago, we took the union of the available simple taxonomy and its relational schema as the ontology. More details of our experiments are publicly available.\(^1\)

**Experiment Setup.** To demonstrate the benefits of our method (Abstraction) presented in Alg. 1, we compare it with respect to the number of detected inconsistency explanations against the following baselines: module-based implementation as described in Section 3.1 (Modular) and the off-the-shelf reasoner [29] (Pellet), in which a KG is processed as a whole. Note that for computing explanations, Pellet is as efficient as other state-of-the-art reasoners, e.g., HermiT [8]. Table 2 presents the sizes of the test ontologies and of the KGs, and the number of entities, classes and properties.

All experiments were performed on a 48 cores machine with 500GB of memory. Pellet reasoner was invoked for computing inconsistency explanations in all settings. In the Abstraction setting, we regard each abstraction generated for a maximal local type (described in Algorithm 1) as a module. We set a timeout of 72 hours for the overall computation and 2 minutes for processing every module in the Abstraction and Modular settings.

**Results.** In Table 3, we compare the three methods with respect to the total number of modules computed, the number of modules processed within the timeout and the number of found explanation patterns together with the actual explanations that they yield.

\(^1\)https://github.com/borschresearch/kg_inconsistency_explanation

---

**Algorithm 1: Computing explanations for inconsistency of a knowledge graph \( G \) w.r.t. an ontology \( O \).**

**Input:** A knowledge graph \( G \) and an ontology \( O \).

**Output:** The set \( \text{allExpls} \) of all explanations for inconsistency of \( G \cup O \).

1. \( \text{allExpls} \leftarrow \emptyset \)
2. \( \text{types} \leftarrow \{\text{ind}(G) \mid a \in \text{ind}(G)\} \)
3. foreach maximal \( \tau \in \text{types} \)
   a. \( X \leftarrow \) all explanations for inconsistency of \( \text{abs}(\tau) \cup O \)
   b. \( X \leftarrow \) obtain the explanations for \( \text{abs}(\tau) \cup O \)
4. foreach \( E \in \text{allExpls} \cup O \in X \) do
   a. \( r' = \text{real}_{\tau'}(v_\ell, E) \)
   b. \( \text{newExpls} \leftarrow \) all realizations of \( r' \) in \( G \)
5. \( \text{allExpls} \leftarrow \text{allExpls} \cup \text{newExpls} \)
6. return \( \text{allExpls} \)

---

**Table 2: Statistics of the datasets.**

<table>
<thead>
<tr>
<th>KG</th>
<th>Axioms</th>
<th>Assertions</th>
<th>Entities</th>
<th>Classes</th>
<th>Relations</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPD</td>
<td>678</td>
<td>929,710</td>
<td>264,081</td>
<td>343</td>
<td>142</td>
</tr>
<tr>
<td>Yago</td>
<td>1,045</td>
<td>7,321,308</td>
<td>3,275,593</td>
<td>960</td>
<td>38</td>
</tr>
<tr>
<td>DBpedia</td>
<td>4,287</td>
<td>22,955,173</td>
<td>5,867,913</td>
<td>685</td>
<td>663</td>
</tr>
</tbody>
</table>

**Table 3: Results of inconsistency explanation computation.**

<table>
<thead>
<tr>
<th>KG</th>
<th>Method</th>
<th>Total</th>
<th>Total</th>
<th>Processed</th>
<th>Patterns</th>
<th>Explanations</th>
<th>Time (hours)</th>
</tr>
</thead>
<tbody>
<tr>
<td>NPD</td>
<td>Pellet</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>41,128</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>Modular</td>
<td>264,081</td>
<td>243,467</td>
<td>-</td>
<td>-</td>
<td>60,554</td>
<td>4.5</td>
</tr>
<tr>
<td></td>
<td>Abstraction</td>
<td>11,970</td>
<td>11,866</td>
<td>20,251</td>
<td>60,554</td>
<td>20,093,617</td>
<td>21</td>
</tr>
<tr>
<td>Yago</td>
<td>Pellet</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>1,547</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>Modular</td>
<td>5,275,593</td>
<td>1,624,196</td>
<td>-</td>
<td>-</td>
<td>73,152</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>Abstraction</td>
<td>2,821</td>
<td>2,821</td>
<td>69</td>
<td>3,565</td>
<td>20,093,617</td>
<td>21</td>
</tr>
<tr>
<td>DBpedia</td>
<td>Pellet</td>
<td>1</td>
<td>0</td>
<td>-</td>
<td>-</td>
<td>72</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>Modular</td>
<td>5,867,913</td>
<td>647,443</td>
<td>-</td>
<td>-</td>
<td>73,152</td>
<td>72</td>
</tr>
<tr>
<td></td>
<td>Abstraction</td>
<td>91,084</td>
<td>90446</td>
<td>1,797</td>
<td>3,565</td>
<td>20,093,617</td>
<td>21</td>
</tr>
</tbody>
</table>
Table 4: Top frequent inconsistency explanation patterns.

<table>
<thead>
<tr>
<th>Abstract KG Assertions</th>
<th>Ontology Axioms</th>
<th>Examples</th>
<th>#Explanations</th>
</tr>
</thead>
<tbody>
<tr>
<td>Yago</td>
<td>DisjointClasses(wn:building, GeoEntity)</td>
<td>wn:building(&lt;J_Forrestal_Building&gt;)</td>
<td>1,768</td>
</tr>
<tr>
<td></td>
<td>ObjectPropertyRange(isLocatedIn, GeoEntity)</td>
<td>isLocatedIn(&lt;NNSA&gt;, &lt;J_Forrestal_Building&gt;)</td>
<td></td>
</tr>
<tr>
<td>DBpedia</td>
<td>SubClassOf(Ship, MeanOfTransportation)</td>
<td>instrument(&lt;African_blues&gt;, &lt;Djembe&gt;)</td>
<td>11,357,430</td>
</tr>
<tr>
<td></td>
<td>SubClassOf(Instrument, Product), SubClassOf(Product, Ship)</td>
<td>bandMember(&lt;Baka_Beyond&gt;, &lt;Djembe&gt;)</td>
<td></td>
</tr>
</tbody>
</table>

**Figure 4:** Distribution of the top-10 frequent explanations.

*Pellet* reasoner failed to process all of the test KGs within the given timeout, i.e., no inconsistency explanations have been computed by this baseline method. Naively splitting the KG into smaller modules is already beneficial yielding some explanations. However, as the number of modules to be considered is still very large, many of them cannot be processed within the time limit, and hence, still only few inconsistency explanations are detected.

The *Abstraction* method significantly reduces the number of modules to be considered to 0.1%, 1.55%, and 4.5% of the original modules (i.e., computed by the *Modular* method) for Yago, DBpedia and NPD respectively. Although not all of the modules could be handled by the invoked reasoner, we managed to process a much larger fraction of the KGs compared to the baselines.

Inconsistency explanations were computed by our method within 6 minutes, 4.5 hours, and 21 hours for Yago, NPD and DBpedia respectively, which naturally correlates with the size and complexity of the considered KGs and their ontologies. The results demonstrate that our method can process large KGs within reasonable times, which are beyond of what earlier proposed approaches can handle.

Figure 4 depicts the top-10 most frequent explanation patterns (X-axis), and the percentage of concrete inconsistency explanations they represent (Y-axis). The majority of explanations follow one of the 5 prominent patterns. Importantly, the analysis of these patterns may reveal systematic issues in the information extraction process, which makes the patterns valuable on their own. As shown in Table 4, about 50% of explanations result from the fact that the ontology specifies that the range of isLocatedIn is GeoEntity, which is disjoint with building. However, in the KG, there are 1.7K triples connecting some entity to an entity belonging to the building class via the isLocatedIn relation. Similarly, in DBpedia, about 57% of explanations are caused by an entity being an object of triples with relations bandMember and playInstrument, whose domains are Person and Instrument respectively, which are known to be disjoint.

## 5 RELATED WORK

The problem of detecting erroneous information in KGs has recently gained a lot of attention. Proposed approaches can be roughly divided into statistics-based [12, 20–22, 27], logic-based [1, 7, 17, 18, 37], and the combination of the first two groups [21, 32], where constraints are learned (e.g., from edit history [32]) and exploited for inconsistency detection. Among the statistics-based methods, the closest to ours are [22, 27], where faulty triples are identified by exploiting statistical distributions of KG relations and types. Another relevant work from this category [15] introduces a machine learning model to measure the trustworthiness of triples.

Our method falls into the logic-based group, in which constraints [25] or ontologies [2, 16] are utilized to identify and repair errors in KGs. In particular, the works [1, 13, 33] are the most relevant; however, in contrast to our method they do not exploit module-based splitting of the KG and the abstractions for inconsistency explanation computation, and do not scale as well as our method.

The idea of using compressed/summarized data and modules to address the scalability issue in large KGs has been explored in several works [4, 35]. Our method is similar in spirit to the abstraction refinement approach [9–11] developed for the purpose of KG expansion, which is different from our goal of computing inconsistency explanations. Importantly, we have identified a new fragment of OWL in which the refinement phase is not necessary and have shown that only maximal local types need to be considered for computing explanations. Computing inconsistency explanations is also done via caching [28], but only limited types of explanations can be discovered, e.g., the most frequent explanations reported in Section 4 have not been detected by the method from [28].

## 6 CONCLUSIONS

We presented a novel method for computing inconsistency explanations of KGs based on KG abstraction. The applicability of our explanations is manifold. For instance, they can be used to facilitate error resolution in KGs e.g., following the automatic approach of Bivenenu et al. [1] or that of Tanon et al. [32], where a human decides which axioms are to be deleted. Another advantage of our method is that it provides error patterns which could reveal systematic issues in the KG construction process. We plan to exploit this direction in the future e.g., by exploiting inconsistency explanations to guide the information extraction process.

Another future direction is to extend our method to handle more expressive ontologies. Currently, it is restricted to the presented OWL 2 fragment. Adding other standard OWL 2 axioms such as sameAs or unrestricted form of someValuesFrom/allValuesFrom axioms, would make the approach incomplete. A possibility for extension of our algorithm to larger fragments of OWL 2 is by adapting it in a "pay-as-you-go" fashion, e.g., by performing additional reasoning steps to handle the new axioms and employing the refinement phase as done in [11].
REFERENCES


