Computing Incoherence Explanations for Learned Ontologies

Daniel Fleischhacker[1], Christian Meilicke[1], Johanna Völker[1]⋆, and Mathias Niepert[2]

[1] Data & Web Science Research Group, University of Mannheim, Germany
{daniel,christian,johanna}@informatik.uni-mannheim.de

[2] Computer Science & Engineering, University of Washington
mniepert@cs.washington.edu

Abstract. Recent developments in ontology learning research have made it possible to generate significantly more expressive ontologies. Novel approaches can support human ontology engineers in rapidly creating logically complex and richly axiomatized schemas. Although the higher complexity increases the likelihood of modeling flaws, there is currently little tool support for diagnosing and repairing ontologies produced by automated approaches. Off-the-shelf debuggers based on logical reasoning struggle with the particular characteristics of learned ontologies. They are mostly inefficient when it comes to detecting modeling flaws, or highlighting all of the logical reasons for the discovered problems. In this paper, we propose a reasoning approach for discovering unsatisfiable classes and properties that is optimized for handling automatically generated, expressive ontologies. We describe our implementation of this approach, which we evaluated by comparing it with state-of-the-art reasoners.

1 Motivation

Ontology learning [2], i.e., automatic generation or enrichment of ontologies, enables ontology engineers to draft both huge and logically complex ontologies within hours or even minutes, thus reducing the costs of ontology development projects to a minimum. However, ontologies generated in a fully automatic way are often flawed containing various (though mostly systematic) types of modeling errors. The greater the size and complexity of a learned ontology, the more important are automatic means to support its inspection and revision by a human ontology engineer. They should point the ontology engineer to as many modeling errors as possible, and ideally suggest suitable fixes.

Though most ontology learning approaches only generate lightweight ontologies, recently, more and more approaches are also able to generate expressive ontologies [11, 17]. While it is reasonable to assume that building an expressive, i.e., richly axiomatized ontology is more error-prone than creating a lightweight taxonomy, a rich axiomatization can also redound to our advantage. This is because the more expressive an ontology is, the more likely it is that modeling errors become manifest as undesired logical consequences which can be detected in an automatic way. Typical examples

⋆ Johanna Völker is financed by a Margarete-von-Wrangell scholarship of the European Social Fund (ESF) and the Ministry of Science, Research and the Arts Baden-Württemberg.

are unsatisfiable classes or properties. These logical consequences can be understood as a symptom for an underlying modeling error that is caused by a set of axioms that contains at least one incorrect axiom. Such combinations of axioms are called an *explanation* for the consequence. Diagnosing modeling errors by computing these explanations is a first step in the overall debugging process, that can be conducted in an automatic [14] or interactive way [9]. Several approaches for computing one or all of the possible explanations have been proposed [8, 10]. However, our experiments show that debugging tools for computing explanations struggle with learned ontologies. This is because learned ontologies commonly share some characteristics which distinguish them from most manually built ontologies:

Redundancy Ontology learning methods often generate logically redundant axioms. Thus, we will find lots of different explanations for the same defect in an ontology. In many cases this is unavoidably entrained by the fact that, for efficiency reasons, logical inference cannot be performed during the learning process. Most often, however, redundancy is a desirable property of ontology learning results, and redundant axioms are generated on purpose to facilitate globally optimal revision at a later stage. A learned ontology will thus contain a relatively high number of axioms compared to a relatively low number of classes and properties. On the contrary, a manually modeled ontology about a similar domain might contain a significantly lower number of axioms.

Restricted Expressivity The result of an ontology learning process usually comprises specific types of axioms, while other types are excluded. Axioms, especially in ontologies generated from textual sources, are more likely to relate named classes than complex class descriptions. Knowing these specifics can be exploited in the reasoning process that is required for computing explanations. Opposed to this, OWL 2 Reasoners, for example Pellet [16] or Hermit [5], are designed to deal with complex class description. Thus, they apply sophisticated methods to deal with a high logical expressivity implementing diverse optimization techniques. However, for computing all (or many) explanations it is required to switch off many optimization techniques in order to trace all explanations for a consequence [16].

In order to address these problems, and to more efficiently spot unwanted logical consequences in learned ontologies, we developed a novel and robust approach for computing explanations. Our approach is applicable to both manually engineered and automatically generated ontologies, however, it has been optimized for the latter. In particular, our approach can be applied to ontologies that contain subsumption and disjointness axioms between named classes and properties, domain and range restrictions, as well as inverse properties. More details on the supported expressivity can be found in Section 4. Our approach is built on a set of rules that is applied to compute consequences from the axioms stated in the ontology. We extend this reasoning approach to compute explanations for unsatisfiable classes and properties. These are the main contributions of the paper.

- We present an approach to computing explanations for unsatisfiable classes and properties based on completion rules. We define a set of completion rules, and we explain how these rules can be used to compute explanations.

- We implemented our approach in a prototype called TRex. TRex is an acronym for "**T**erminological **R**easoning with **Ex**planations". The TRex code, as well as the datasets used in our experiments, is publicly available as open source.[3]
- In our experiments, we applied Pellet and TRex to compute unsatisfiabilities and their explanations. Our results show that TRex can compute explanations for the ontologies used in our experiments, while Pellet fails to generate more than one explanation for each unsatisfiability.
- We analyze the limits of our approach and describe ways for improving and extending our prototype.

The remaining parts of the paper are structured as follows. We first give an overview of related work (cf. Section 2). In Section 3, we formally define the notions of unsatisfiability and incoherence and introduce the notion of an explanation. In Section 4, we describe the approach implemented in TRex. In Section 5, we report on our experiments and present the results. Finally, we conclude with an outlook to future work (cf. Section 6) and a summary (cf. Section 7).

2 Related Work

As already argued, incoherence can point to erroneous axioms, or at least to sets of axioms that explain a certain defect. To exploit this in an automated setting is especially important in the case of ontology learning. An example can be found in the work of Meilicke et al. [13] where debugging techniques have been applied to ensure the coherence of the learned ontology. In the proposed greedy-approach, learned axioms are added step by step, each time checking the coherence of the learned ontology. This approach avoids the computation of explanations and the optimality of the debugging result cannot be guaranteed. Lehmann and Bühmann [12] have proposed a tool for repairing and enriching knowledge bases called ORE. Aside from the learning parts, ORE identifies root unsatisfiable classes and computes explanations for a chosen unsatisfiable class. The system description indicates that ORE uses Pellet as underlying reasoner. The results presented in [12] focus on relatively small ontologies and do not include runtime statistics. Both aspects are in the focus of our experiments. Further work in this direction has been made by Völker and Rudolph [18] and Haase and Völker [7].

Pellet [16] is a state-of-the-art OWL 2 reasoner that offers additional methods for computing explanations. It can thus be used to compute explanations for unsatisfiabilities. The explanation component of Pellet is based on a glass-box approach that is built on the tableau-based decision procedures of Pellet. As shown by [10] a glass-box approach outperforms black-box techniques. Since Pellet is used by a large community, implements (most of) the required functionality, and can be used without any additional setup, we report on its performance within the experimental section. Note that the technique that we propose is not based on a tableau-based procedure, but uses completion rules to materialize all entailments.

Qi et al. [14] have focussed on both the theoretical foundations and on the algorithms of computing a diagnosis. The authors report on experiments using data from

[3] `http://dfleischhacker.github.com/trex-reasoner`

the field of ontology learning and ontology mapping. These experiments include the computation of the explanations as well as the strategy of resolving the incoherence by removing some axioms from the explanations. However, the authors re-use the black-box approach proposed by Kalyanpur et.al [10] for finding minimal incoherency preserving subsets. As already mentioned, this algorithm has turned out to be less efficient compared to the glass-box approach implemented in Pellet.

An interesting approach for computing explanations for inconsistencies has recently been proposed by Wu et al. [19]. The authors propose a MapReduce-based approach for distributing the computation of explanations for OWL *pD**. OWL *pD** provides a complete set of entailment rules that is used by the authors. The basic approach is thus similar to the approach that we propose aside from the fact that we do not distribute our approach in the context of a MapReduce framework. However, OWL *pD** does not support disjoint properties, thus, it cannot be applied to our scenario. Moreover, the authors focus on explanations for inconsistencies that occur in comprehensive A-Boxes, while we focus on unsatisfiable classes and properties.

Similar to the approach of Wu et. al, our approach is also based on the idea of a Truth Maintenance System [3]. We use a set of rules to entail new axioms from a set of given axioms keeping track of the dependency tree that emerges during the iterative process. A set of completion rules is defined for the OWL RL profile[4], which is expressive enough to support those ontologies that are in the focus of this work. However, we finally decided to design a weaker rule set, which is better tailored to our needs. In particular, we are not interested in ABox reasoning, nor do we require many constructs supported by OWL RL. On the other hand, the profile of OWL EL[5] is too restrictive for our needs, since it does not support property disjointness and inverse properties.

3 Preliminaries

In the following, we introduce some notions from the field of ontology debugging. First, we start with the notion of unsatisfiability.

Definition 1 (Unsatisfiability and Incoherence). *A class description C is unsatisfiable in an ontology $\mathcal{O}$ iff for each model $\mathcal{I}$ of $\mathcal{O}$ we have $C^{\mathcal{I}} = \emptyset$.*

Due to a common understanding [6], the unsatisfiability of a named class C indicates that some of the axioms in the ontology $\mathcal{O}$ are incorrect. The idea is that a class should be used to specify the type of instances. However, if an unsatisfiable class C is used in a class assertion $C(a)$, this results in the inconsistency of $\mathcal{O}$. We can check whether C is unsatisfiable by asking a reasoner whether $\mathcal{O} \models C \sqsubseteq \bot$ holds. While the classic notion of incoherence is limited to class unsatisfiability, the same line of argumentation holds for properties. The usage of an unsatisfiable property P in a property assertion $P(a, b)$ results in the inconsistency of $\mathcal{O}$. Again, we can use a reasoner to check whether a certain property P is unsatisfiable. This time, we have to check whether the entailment $\mathcal{O} \models \exists P.\top \sqsubseteq \bot$ holds. The following definition completes our definition provided before.

[4] http://www.w3.org/TR/owl2-profiles/#OWL_2_RL

[5] http://www.w3.org/TR/owl2-profiles/#OWL_2_EL

Definition 1 (continued). *A property description P is unsatisfiable in an ontology $\mathcal{O}$ iff for each model $\mathcal{I}$ if $\mathcal{O}$ we have $P^{\mathcal{I}} = \emptyset$. An ontology that contains an unsatisfiable named class or property is called an incoherent ontology.*

The unsatisfiability of classes or properties, and thus the incoherence of an ontology, can be traced back to their root causes. These causes are the minimal sets of axioms, which are called explanation or justification, being strong enough to entail the unsatisfiability or more general a specific axiom holding in the ontology. According to Kalyanpur et al. [10], an explanation is formally defined as follows.

Definition 2 (Explanation). *Given an ontology $\mathcal{O}$ and an axiom α, a subset $\mathcal{O}' \subseteq \mathcal{O}$ is an explanation for α iff $\mathcal{O}' \models \alpha$ and there exists no $\mathcal{O}'' \subset \mathcal{O}'$ such that $\mathcal{O}' \models \alpha$.*

Note that a certain axiom can have many different explanations, because different minimal subsets of the axioms in the learned ontology allow to entail this axiom. In the following example, the object property *secondDriverCountry* is unsatisfiable with its unsatisfiability caused by two overlappings explanations. Note that this example is taken from the ontologies used in our experiments in Section 5.

Ontology:
$\mathit{Country} \sqsubseteq \neg \mathit{Settlement}$
$\mathit{secondDriverCountry} \sqsubseteq \mathit{location}$
$\mathit{secondDriverCountry} \sqsubseteq \neg \mathit{location}$
$\top \sqsubseteq \forall \mathit{secondDriverCountry}.\mathit{Country}$
$\top \sqsubseteq \forall \mathit{location}.\mathit{Settlement}$

Unsatisfiable Property:
secondDriverCountry

Explanation 1:
$\mathit{Country} \sqsubseteq \neg \mathit{Settlement}$
$\mathit{secondDriverCountry} \sqsubseteq \mathit{location}$
$\top \sqsubseteq \forall \mathit{secondDriverCountry}.\mathit{Country}$
$\top \sqsubseteq \forall \mathit{location}.\mathit{Settlement}$

Explanation 2:
$\mathit{secondDriverCountry} \sqsubseteq \mathit{location}$
$\mathit{secondDriverCountry} \sqsubseteq \neg \mathit{location}$

4 Approach

In this section, we present our approach to computing explanations for unsatisfiable classes and properties. Our approach is applicable to the OWL 2 fragment that is defined by the axiom types listed in the leftmost column of Table 1. Note that all classes and properties that appear in Table 1 are named classes and properties.

We use the notation shown in the second column of Table 1, i.e., we assign a first-order predicate symbol to each type of axiom, to distinguish between entailments derived by applying our completion rules and axioms that hold in the ontology due to the standard model-theoretic semantics. Our approach is based on rules (1) to (24). Given an ontology $\mathcal{O}$, we start with an initial set of formulae that are equivalent to the axioms stated in $\mathcal{O}$. Then, we iteratively apply the set of rules to all stated and derived formulae until no further formula can be derived. We refer to the resulting set of all derived formulae as $\mathbf{E}_{\mathcal{O}}$. If $cdis(A, A) \in \mathbf{E}_{\mathcal{O}}$ or $pdis(P, P) \in \mathbf{E}_{\mathcal{O}}$, we conclude that class A or property P, respectively, is unsatisfiable. If there exists no such A or P, we conclude that $\mathcal{O}$ contains no unsatisfiable class, which means that $\mathcal{O}$ is coherent.

Table 1. Types of supported axioms.

Type of axiom	**First-order predicate symbol**	**Description**
$A \sqsubseteq B$	$csub(A, B)$	Class Subsumption
$P \sqsubseteq Q$	$psub(P, Q)$	Property Subsumption
$A \sqsubseteq \neg B$	$cdis(A, B)$	Class Disjointness
$P \sqsubseteq \neg Q$	$pdis(P, Q)$	Property Disjointness
$\exists P.\top \sqsubseteq A$	$dom(P, A)$	Domain Restriction
$\top \sqsubseteq \forall P.A$	$range(P, A)$	Range Restriction
$P^{-1} \sqsubseteq Q$	$psubinv(P, Q)$	Inverse Property Subsumption
$P^{-1} \sqsubseteq \neg Q$	$pdisinv(P, Q)$	Inverse Property Disjointness

$$\Rightarrow csub(A, A) \tag{1}$$

$$cdis(A, B) \Rightarrow cdis(B, A) \tag{2}$$

$$csub(A, B), csub(B, C) \Rightarrow csub(A, C) \tag{3}$$

$$csub(A, B), cdis(B, C) \Rightarrow cdis(A, C) \tag{4}$$

$$\Rightarrow psub(P, P) \tag{5}$$

$$pdis(P, Q) \Rightarrow pdis(Q, P) \tag{6}$$

$$psub(P, Q), psub(Q, R) \Rightarrow psub(P, R) \tag{7}$$

$$psub(P, Q), pdis(Q, R) \Rightarrow pdis(P, R) \tag{8}$$

$$dom(P, A), csub(A, B) \Rightarrow dom(P, B) \tag{9}$$

$$ran(P, A), csub(A, B) \Rightarrow ran(P, B) \tag{10}$$

$$psub(P, Q), dom(Q, A) \Rightarrow dom(P, A) \tag{11}$$

$$psub(P, Q), ran(Q, A) \Rightarrow ran(P, A) \tag{12}$$

$$cdis(A, B), dom(P, A), dom(P, B) \Rightarrow pdis(P, P) \tag{13}$$

$$cdis(A, B), ran(P, A), ran(P, B) \Rightarrow pdis(P, P) \tag{14}$$

$$psubinv(P, Q), dom(Q, A) \Rightarrow ran(P, A) \tag{15}$$

$$psubinv(P, Q), ran(Q, A) \Rightarrow dom(P, A) \tag{16}$$

$$psubinv(P, Q), psubinv(Q, R) \Rightarrow psub(P, R) \tag{17}$$

$$psubinv(P, Q), psub(Q, R) \Rightarrow psubinv(P, R) \tag{18}$$

$$psub(P, Q), psubinv(Q, R) \Rightarrow psubinv(P, R) \tag{19}$$

$$pdisinv(P, Q), psub(R, Q) \Rightarrow pdisinv(P, R) \tag{20}$$

$$psubinv(P, Q), pdis(Q, R) \Rightarrow pdisinv(P, R) \tag{21}$$

$$psubinv(P, Q), pdisinv(Q, R) \Rightarrow pdisinv(P, R) \tag{22}$$

$$pdisinv(P, Q) \Rightarrow pdisinv(Q, P) \tag{23}$$

$$pdisinv(P, P) \Rightarrow pdis(P, P) \tag{24}$$

Given the set of rules, there are two important questions. The first question is related to the *correctness* of these rules and the second question is related to the *completeness*.

The correctness of each single rule follows directly from the standard DL semantics. Suppose that we derive a formula $cdis(A, A) \in \mathbf{E}_{\mathcal{O}}$. This also means that $\mathcal{O} \models A \sqsubseteq \neg A$ and thus $A^{\mathcal{I}} = \{\}$ for each interpretation $\mathcal{I}$. The same holds for properties. We conclude that our approach is sound with respect to computing entailments, and thus also sound with respect to detecting unsatisfiable classes and properties.

With respect to the second question, we first need to show that the following proposition holds. The proof for this proposition is available in a technical report.[6]

Proposition 1. *If $\mathcal{O}$ is incoherent, there exists a class A with $cdis(A, A) \in \mathbf{E}_{\mathcal{O}}$ or a property P with $pdis(P, P) \in \mathbf{E}_{\mathcal{O}}$.*

Note that we will finally argue that our reasoner is able to compute all explanations for all unsatisfiable classes and properties. To show this, we need to describe our approach to computing explanations in further detail. As mentioned above, we apply the completion rules iteratively to derive new entailments. This can be conducted in an ordered way to reduce the checking of possible candidates for deriving new formulae. In particular, we proceed as follows with $\mathbf{E}_{\mathcal{O}}$, $\mathbf{E}'_{\mathcal{O}}$, and $\mathbf{E}''_{\mathcal{O}}$ initialized as empty sets.

Class Subsumption We add all formulae $csub(A, B)$ corresponding to stated axioms to $\mathbf{E}'_{\mathcal{O}}$. Then, we add those formulae that are entailed by rule (1) to $\mathbf{E}'_{\mathcal{O}}$. We apply rule (3) on $\mathbf{E}'_{\mathcal{O}}$ until we cannot derive new formulae. Since no $csub(A, B)$ appears in the head of any other rule, we know that $\mathbf{E}'_{\mathcal{O}}$ is $csub$ saturated.

Property Subsumption We add all formulae $psub(A, B)$ and $psubinv(A, B)$ corresponding to stated axioms to $\mathbf{E}''_{\mathcal{O}}$. Then, we add those formulae that are entailed by rule (5) to $\mathbf{E}''_{\mathcal{O}}$. We apply rules (7), (17), (18), and (19) on $\mathbf{E}''_{\mathcal{O}}$ until we cannot derive new formulae. Since there appears no $psub(P, Q)$ or $psubinv(P, Q)$ in the head of any other rule, we know that $\mathbf{E}''_{\mathcal{O}}$ is $psub$ and $psubinv$ saturated.

Domain and Range We set $\mathbf{E}_{\mathcal{O}} = \mathbf{E}'_{\mathcal{O}} \cup \mathbf{E}''_{\mathcal{O}}$. We add all formulae $dom(P, A)$ and $ran(P, B)$ corresponding to stated axioms to $\mathbf{E}_{\mathcal{O}}$. We apply rules (9), (10), (11), (12), (15), and (16) on $\mathbf{E}_{\mathcal{O}}$ until we cannot derive new formulae. Since no $dom(P, A)$ or $ran(P, B)$ appears in the head of any other rule, we know that $\mathbf{E}_{\mathcal{O}}$ is dom and ran saturated.

Class Disjointness We add all formulae $cdis(A, B)$ corresponding to stated axioms to $\mathbf{E}_{\mathcal{O}}$. We apply rule (2) and (4) on $\mathbf{E}_{\mathcal{O}}$ until we cannot derive new formulae. Since there appears no $cdis(A, B)$ in the head of any other rule, we know that $\mathbf{E}_{\mathcal{O}}$ is $cdis$ saturated.

Property Disjointness We add all formulae $pdis(P, A)$ and $pdisinv(P, B)$ corresponding to stated axioms to $\mathbf{E}_{\mathcal{O}}$. We apply all remaining rules until we cannot derive new formulae. $\mathbf{E}_{\mathcal{O}}$ is now saturated with respect to all types of formulae.

If we stop the entailment process as soon as it is not possible to derive any new entailment, it will not be possible to compute all explanations. Thus, we have to use a different criterion for moving from one step to the next and finally for terminating the whole process. The idea is to continue with the next step (or to terminate) only if there exists no $\alpha \in \mathbf{E}_{\mathcal{O}}$ such that the explanation of α has been modified during the last iteration. Let now $expl(\alpha)$ denote the set of all explanations for a given formula α that is added to $\mathbf{E}_{\mathcal{O}}$ during executing the process described above. For the sake of simplicity,

[6] http://dfleischhacker.github.com/trex-reasoner

we only mention $\mathbf{E}_{\mathcal{O}}$ in the following, which might refer to $\mathbf{E}_{\mathcal{O}}$, $\mathbf{E}'_{\mathcal{O}}$ or $\mathbf{E}''_{\mathcal{O}}$ depending on the current phase of the process. We have to distinguish between two cases.

- α corresponds to a stated axiom in $\mathcal{O}$. We set $expl(\alpha) = \{\{\alpha\}\}$.
- α is derived by one of the other rules. We set $expl(\alpha) = expl(\alpha) \cup \{\{expl(\beta_1), \ldots, expl(\beta_n)\}\}$ where $\beta_1, \ldots, \beta_n$ refers to those formulae that triggered the rule.

Due to the recursive character of an explanation, $expl(\alpha)$ can be understood as a disjunction of conjunctions, that might again be built from a disjunction of conjunctions, and so forth. Thus, the approach, as it has been described so far, constructs an or-and-tree of explanations. However, we want to avoid the construction of a complex tree by ensuring that $expl(\alpha)$ is always stored as a DNF, i.e., $expl(\alpha)$ is always a disjunction of conjunctive clauses. To guarantee the explanations to be in DNF, we apply the distributivity law every time we combine explanations. Afterwards, we minimize the resulting DNF by removing conjunctions that are supersets or duplicates of other conjunctions. Checking for duplicates is important with respect to our termination criteria, because a DNF to which we try to add a duplicate or a superset should not be counted as an explanation that has been modified.

Now, we show that our approach computes all minimal incoherence preserving subsets of an incoherent ontology $\mathcal{O}$. Schlobach and Cornet [15] have defined a MIPS M (minimal incoherence preserving TBox) as a subset $M \subseteq \mathcal{O}$ such that M is incoherent and each $M' \subset M$ is coherent. An explanation of an unsatisfiable class (or property) is called a MUPS (minimal unsatisfiability preserving TBox) in the terminology of Schlobach and Cormet. Given that, each MUPS is a MIPS or a superset of a MIPS. Let now $MIPS(\mathcal{O})$ refer to the set of all MIPS in an incoherent ontology $\mathcal{O}$. Furthermore, let now $expl_u(\mathcal{O})$ refer to the union of explanations for unsatisfiable classes or properties that are computed by our approach.

In the following, we prove that $MIPS(\mathcal{O}) \subseteq expl_u(\mathcal{O})$. For that proof, we have to take into account that our approach is monotonic in the sense that $expl_u(\mathcal{O}) \supseteq expl_u(\mathcal{O}')$ if $\mathcal{O} \supseteq \mathcal{O}'$. This follows from the fact that we apply the completion rules unless no additional explanation can be added. Thus, if $\mathcal{O}$ is a superset of $\mathcal{O}'$ we will never compute fewer explanations for $\mathcal{O}$ than for $\mathcal{O}'$. Let us now apply our method to each $M \in MIPS(\mathcal{O})$. Each M is by definition an incoherent ontology. According to Proposition 1, we will thus at least detect one unsatisfiable class or property for M. Since the computation of the unsatisfiable class (or property) is, within our approach, directly coupled to the computation of an explanation, we will always compute an explanation with respect to M, i.e., $expl_u(M) \neq \{\}$. Since M is a MIPS, there exists no incoherent subset M' of M. Thus, we end up with $expl_u(M) = M$. Further, we know that $M \subseteq \mathcal{O}$ and thus we conclude, based on the monotonicity of our approach, that $expl_u(M) \subseteq expl_u(\mathcal{O})$. We conclude that $MIPS(\mathcal{O}) = \bigcup_{M \in MIPS(O)} expl_u(M) \subseteq expl_u(O)$.

We have thus shown that our approach detects all explanations for unsatisfiable classes and properties, as long as those explanations are not subsets of other explanations. With respect to exploiting explanations in a debugging context, it is thus not important to keep track of those explanations for which we have not yet proven that we are able to detect them. If we apply an algorithm for resolving all unsatisfiabilities on $expl_u(\mathcal{O})$, this algorithm will always (implicitly) resolve all unsatisfiabilities.

Implementation We implemented this approach in a prototype that is mainly based on a matrix representation for each type of formula. We define, for example, a boolean matrix for all formulae $csub(X, Y)$, where X is associated to a row and Y is associated to a column in the matrix. We first initialize the matrix with all stated axioms. Then we apply rule (1) adding entries to the diagonal of this matrix. The set of entailments $\mathbf{E}_{\mathcal{O}}$ corresponds to the entries in our matrix representation. The cell (X, Y) also points to the set of explanations $expl(csub(X, Y))$. We have chosen a similar representation for all other types of formulae. After initializing all matrices, we apply the rules as described above to entail new entries in the matrices and to update the corresponding explanations. The diagonal of the matrices for predicates $cdis$ and $pdis$ finally refers to the set of unsatisfiable classes and properties and their corresponding explanations.

Note again, that we have developed the approach for debugging ontologies that have been learned automatically. Learned ontologies will typically contain subsumption axioms between most pairs of classes that subsume each other, even though most of these axioms can be derived from other axioms that have also been learned. The same holds for disjointness axioms. Thus, we expect that most matrices for learned ontologies are dense or not as sparse as matrix representations of carefully modeled ontologies. In such a setting using a matrix representation is less critical with respect to memory and runtime issues as it will be the case in other scenarios.

Finally, we have not yet implemented support for inverse properties in the current prototype, i.e., rules (15) to (24) are still missing from the implementation. This is because the dataset that we used for our experiments, contains only a small number of axioms that involved inverse properties. An implementation based on matrices is thus not well-suited in terms of efficiency, and further improvements beyond the scope of this paper would be required to make this type of approach feasible in practice. Therefore, we decided to remove all inverse properties axioms from the datasets that we used in the experiments presented in the following section.

5 Experiments

5.1 Setting

The ontologies used in our experiments are based on the ontology that has been created by the learning approach described in [4] and [17], respectively. This ontology is based on the original DBpedia ontology [1] and has been enriched by means of statistical schema induction on the DBpedia instance data. For analyzing the impact of different ontology sizes on the reasoning performance, we created subsets from this full ontology starting with a base ontology containing randomly selected 20% of the total number of axioms. We gradually added randomly selected and not yet contained axioms from the full ontology. While growing the base ontology, we regularly took snapshots resulting in a set of 11 ontologies $\mathcal{O}_0$ to $\mathcal{O}_{10}$ where each ontology $\mathcal{O}_i$ is a subset of $\mathcal{O}_{i+1}$. The last snapshot is equivalent to the full ontology. Statistics about these ontologies, which fall all into the $\mathcal{ALCH}$ expressivity class, can be found in Table 2.

Table 2. Statistics about ontologies used in experiments.

Ontology	Axioms	Classes	Properties	Unsat. Classes	Unsat. Properties
$\mathcal{O}_0$	23,706	300	654	3	5
$\mathcal{O}_1$	32,814	304	673	6	7
$\mathcal{O}_2$	41,941	309	689	9	14
$\mathcal{O}_3$	51,056	316	702	15	29
$\mathcal{O}_4$	60,166	319	714	26	50
$\mathcal{O}_5$	69,271	321	724	32	82
$\mathcal{O}_6$	78,375	323	730	49	112
$\mathcal{O}_7$	87,468	324	736	63	162
$\mathcal{O}_8$	96,555	324	737	83	209
$\mathcal{O}_9$	105,642	324	742	132	336
$\mathcal{O}_{10}$	114,726	324	742	152	396

We consider two use cases which are relevant for debugging automatically generated ontologies. The first use case is the detection of unsatisfiable classes and properties. The second use case is based on the detection step and deals with finding explanations for discovered unsatisfiabilities. We compared TRex with two state-of-the-art reasoners, Pellet and Hermit. While TRex fully supports both use cases, Hermit and Pellet are only able to handle sub sets. Regarding the first use case, Hermit[7] provides direct programmatic access to the set of unsatisfiable classes. The retrieval of unsatisfiable properties is not directly possible. Instead, we resort to retrieving all sub properties of `owl:bottomObjectProperty`. Hermit does not provide support for generating explanations, so it is not suited for our second use case. The Pellet reasoner[8] also supports direct retrieval of unsatisfiable classes. In contrast to Hermit, it does not support retrieving subproperties of `owl:bottomObjectProperty`. Thus, we are not able to directly retrieve unsatisfiable properties with Pellet without further modifications.

A feature which distinguishes Pellet from Hermit is the support for computing explanations. To have the possibility to compare our explanation results with other explanations, we implemented a way of reducing the detection of unsatisfiable properties to the detection of unsatisfiable classes. We extended the ontologies with the axiom $C_P \sqsubseteq \exists P.\top$ for each object property P in the ontology where C_P is a fresh class introduced for the respective property. Based on this, we know that C_P is unsatisfiable iff P is unsatisfiable. In our experiments, this variant of Pellet is referred to as *PelletMod*.

5.2 Results

The result of the first use case are depicted in Table 3.[9] All reasoners discovered the same number of unsatisfiabilities except for Pellet because of its inability to detect un-

[7] `http://www.hermit-reasoner.com`, Version 1.3.6

[8] `http://clarkparsia.com/pellet`, Version 2.3.0

[9] All experiments have been conducted on a Quad-core Intel Core i7 with 3.07GHz and 24GB RAM. The results are averaged over 5 runs.

satisfiable properties. Overall, Hermit is the fastest reasoner for retrieving the set of all unsatisfiable classes and properties. In particular, Hermit provides the best scalability in our experiments since the runtime behaviour is second to none of the other reasoners. The runtimes of Pellet and PelletMod increase much more with respect to the ontology size. The runtimes of TRex are the highest for all ontology sizes. TRex is designed to always determine explanations for all inferable axioms. Pellet only computes explanations if those are explicitly requested for specific axioms. Furthermore, TRex has higher initialization costs. However, these initialization costs are hardly affected by the growing number of axioms.

Table 3. Runtimes in milliseconds for the detection of unsatisfiabilities.

Ontology	Pellet	PelletMod	Hermit	TRex
$\mathcal{O}_0$	392	411	450	6,630
$\mathcal{O}_1$	621	654	629	7,169
$\mathcal{O}_2$	910	997	720	7,839
$\mathcal{O}_3$	1,232	1,297	849	8,425
$\mathcal{O}_4$	1,485	1,854	1,916	9,889
$\mathcal{O}_5$	1,970	2,088	1,158	9,411
$\mathcal{O}_6$	2,419	2,617	1,295	9,572
$\mathcal{O}_7$	2,897	3,063	1,468	12,559
$\mathcal{O}_8$	3,460	3,585	1,549	10,124
$\mathcal{O}_9$	3,823	3,899	1,721	11,148
$\mathcal{O}_{10}$	4,327	4,439	1,864	12,006

The results of the second use case are provided in Table 4. These results are the runtimes for retrieving the explanations for each of the unsatisfiable classes and properties found by the respective reasoner. Thus, the runtimes of Pellet only include the explanation retrieval for unsatisfiable classes. An important fact is that Pellet and PelletMod runtimes are only those that we measured for retrieving *a single explanation* per unsatisfiability while the TRex runtimes include the retrieval of *all explanations* for all discovered unsatisfiabilities. The number of all explanations as found by TRex is provided in the right-most column. In contrast, Pellet generates one explanation for each unsatisfiable class while PelletMod generates one for each unsatisfiable class or property. We also conducted experiments with Pellet and PelletMod retrieving multiple explanations for each unsatisfiability but the retrieval of multiple explanations in Pellet turned out to be highly unstable for the ontologies used in our experiments. We observed in all cases runtime exceptions of Pellet and PelletMod making it impossible to compute more than one explanation. When we tried to catch these exceptions, the runtimes measured were also significantly higher than the runtimes measured for TRex. However, it remained unclear whether those runtimes were caused by the exceptions or by the correct execution of the algorithm until the exception occurred. For that reason we omitted to present these results, which would in any case be based on incomplete sets of explanations, and resorted to single-explanation retrieval for Pellet and PelletMod.

As we see from the given table, the runtime of TRex for retrieving all explanations for all unsatisfiabilities is increasing exponentially with the number of axioms contained

Table 4. Runtimes in milliseconds for generating explanations for unsatisfiable classes and properties. "All Explanations" means all MIPS.

	Single Explanation				**All Explanations**	
	Pellet		**PelletMod**		**TRex**	
Ontology	**Runtime**	**# Expl.**	**Runtime**	**# Expl.**	**Runtime**	**# Expl.**
$\mathcal{O}_0$	848	3	863	8	6,758	8
$\mathcal{O}_1$	1,317	6	1,365	13	7,594	13
$\mathcal{O}_2$	1,899	9	1,956	23	9,011	26
$\mathcal{O}_3$	2,463	15	2,693	44	9,892	54
$\mathcal{O}_4$	3,341	26	3,530	76	11,666	100
$\mathcal{O}_5$	4,070	32	4,322	114	11,732	158
$\mathcal{O}_6$	5,068	49	5,235	161	12,980	250
$\mathcal{O}_7$	5,979	63	6,309	225	17,495	386
$\mathcal{O}_8$	7,082	83	7,396	292	21,726	686
$\mathcal{O}_9$	7,805	132	8,228	468	44,966	2,031
$\mathcal{O}_{10}$	8,947	152	9,480	548	66,781	2,722

in the ontology while the other reasoners only suffer from a linear increase. However, this increase can be explained by means of the aforementioned difference in the number of explanations. While Pellet and PelletMod only retrieve one explanation per unsatisfiability, which means that the total number of explanations is linear to the number of unsatisfiable classes and properties, the total number of explanations retrieved by TRex is not linearly bound but instead growing exponentially in the number of axioms. This is also depicted in Figure 1. In this Figure we plotted both the runtimes of TRex and the number of computed explanations on the y-axis. Again, we observe a relatively high initialization cost. However, at some point in time the runtimes of TRex seem to grow linear in the number of computed explanations.

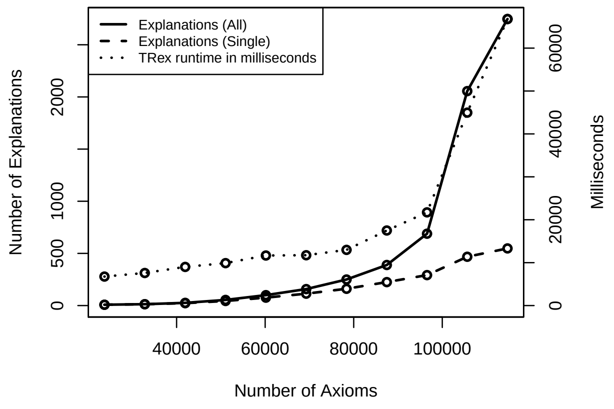

Fig. 1. TRex explanation runtimes and the total number of retrieved explanations.

Explanation Completeness In order to obtain additional evidence for the completeness of the explanation component and the correctness of our implementation, we set up another experiment. We implemented the simple ontology schema debugging approach described in Algorithm 1. Given an incoherent ontology $\mathcal{O}$, this algorithm constructs a randomly chosen minimal hitting H set over all explanations $expl_u(\mathcal{O})$ computed by TRex. Removing H from $\mathcal{O}$ should always result in a coherent ontology. We ran this algorithm 200 times on $\mathcal{O}_{10}$. The computed hitting sets contained between 201 and 223 axioms. After each run, we tested the resulting ontology for coherence using TRex and Hermit. For each run, both reasoners did not find further unsatisfiable classes or properties.

Algorithm 1 Randomized greedy ontology debugging

```
function RandomizedGreedyDebug($O, expl_u(\mathcal{O})$)
    $H \leftarrow \{\}$                                  ▷ set for storing already removed axioms
    for all $e \in expl_u(\mathcal{O})$ do              ▷ $e$ is an explanation, i.e., a set of axioms
        if $e \cap H = \varnothing$ then
            $a \leftarrow$ randomly chosen axiom from $e$
            $O \leftarrow O \setminus \{a\}$
            $H \leftarrow H \cup \{a\}$
```

6 Future Work

The *runtime performance* of TRex is strongly affected by the way of detecting new or changed inferred axioms and their explanations. Currently, we revisit all axioms in each iteration. Especially in later phases, when only a few axioms continue to change, this causes a large overhead. Thus, we will implement a better change tracking combined with appropriate index structures to find resolvable axioms. Such techniques are typically implemented in theorem provers and other engines using inference rules. We also consider implementing large parts of the reasoning process in a database infrastructure. Furthermore, we might investigate the usage of more efficient data structures for managing explanations, like ordered binary decision diagrams. However, for our current use cases, the storage of explanations does not impose limits to our systems.

Currently, TRex does not implement the completion rules for *inverse properties*. The straight-forward implementation in the current structures of TRex would drastically reduce its performance. Once we applied the modifications described in the previous paragraph, we will also implement the support for inverse properties. These modifications are intended to reduce the impact that a small number of axioms involving inverse properties has on the runtime performance.

Subsumption cycles are explanations for $\mathcal{O} \models A \equiv B$ where $A \equiv B \notin \mathcal{O}$. Such an equivalence is not necessarily an undesired consequence, however, it might be worthwhile to analyze the involved axioms. Now suppose that we want to compute subsumption cycles for a highly incoherent ontology where most classes, including two classes A and B, are incoherent. Due to that fact that A and B are subsumed by bottom, we have $\mathcal{O} \models A \equiv B$. The explanations for the unsatisfiability of A and B are thus also responsible for the equivalence of A and B. This is not the case in our approach, because $\bot$ is not included in our formalization. Subsumption cycles are thus not affected by unsatisfiabilities. We will exploit this advantage of our approach in extending our system to support the detection of subsumption cycles.

Furthermore, we plan to use TRex in the future for the purpose it has been designed for, namely, for debugging highly incoherent learned ontologies. In doing so, we will apply different techniques for constructing minimal hitting sets over the set of explanations generated by TRex to improve the overall quality of the learned ontologies.

7 Summary

In this paper, we have presented an approach to computing explanations for unsatisfiable classes and properties in an incoherent ontology. We have shown that our method is complete and sound with respect to computing all minimal incoherence preserving subsets, provided that the input ontology belongs to a certain OWL 2 fragment. While the expressivity supported by our approach is thus limited to a comparatively small subset of OWL 2, we have argued that ontologies generated by state-of-the-art ontology learning approaches are unlikely to exceed the supported level of modeling complexity.

We implemented our approach by developing a prototype called TRex[10]. Besides our theoretical considerations, we have experimentally verified the completeness of our implementation by showing that a debugging approach, which makes use of the computed explanations, always constructs a coherent ontology if applied to the ontology with the highest degree of incoherency from our test set.

Furthermore, the results of our experiments show that TRex successfully handles input ontologies which are challenging for Pellet at least as far as the non-standard reasoning task of computing explanations is concerned. Unlike TRex, Pellet fails to generate more than one explanation for each unsatisfiable class or property. Pellet turned out to be less robust and stable than our implementation if applied to the learned ontologies used in our experiments. Our comparison of Hermit, Pellet, and TRex on the standard reasoning task of computing unsatisfiable classes or properties, indicates that TRex requires significantly more time. However, these differences in terms of runtime can (partially) be explained by the fact that TRex always generates all of the possible explanations. Finally, we identified some potential for improving our prototype, which could increase the performance of future releases.

Overall, we conclude that TRex might turn out to be a valuable and robust tool for debugging learned ontologies. The challenges that we meet in the future will help us to further improve TRex in terms of supported functionality and efficiency.

References

1. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: DBpedia - a crystallization point for the web of data. Web Semantics 7(3), 154–165 (2009)

[10] The source code of this prototype, as well as the ontologies used in our experiments, are available from `http://dfleischhacker.github.com/trex-reasoner`.

2. Cimiano, P., Hotho, A., Staab, S.: Learning concept hierarchies from text corpora using formal concept analysis. Journal of Artificial Intelligence Research 24, 305–339 (AUG 2005)
3. Doyle, J.: A truth maintenance system. Artificial intelligence 12(3), 231–272 (1979)
4. Fleischhacker, D., Völker, J., Stuckenschmidt, H.: Mining RDF data for property axioms. In: On the Move to Meaningful Internet Systems: OTM 2012, Lecture Notes in Computer Science, vol. 7566, pp. 718–735. Springer Berlin Heidelberg (2012)
5. Glimm, B., Horrocks, I., Motik, B., Stoilos, G.: Optimising ontology classification. In: Proc. of the 9th Intl. Semantic Web Conference (ISWC). pp. 225–240 (2010)
6. Haase, P., Qi, G.: An analysis of approaches to resolving inconsistencies in DL-based ontologies. In: Flouris, G., d'Aquin, M. (eds.) Proc. of the International Workshop on Ontology Dynamics. pp. 97–109 (2007)
7. Haase, P., Völker, J.: Ontology learning and reasoning – dealing with uncertainty and inconsistency. In: Uncertainty Reasoning for the Semantic Web I, LNCS, vol. 5327, pp. 366–384. Springer Berlin / Heidelberg (2008)
8. Ji, Q., Qi, G., Haase, P.: A relevance-directed algorithm for finding justifications of DL entailments pp. 306–320 (2009)
9. Jiménez-Ruiz, E., Grau, B.C., Horrocks, I., Llavori, R.B.: Ontology integration using mappings: Towards getting the right logical consequences. In: The Semantic Web: Research and Applications, 6th European Semantic Web Conference. pp. 173–187 (2009)
10. Kalyanpur, A., Parsia, B., Horridge, M., Sirin, E.: Finding all justifications of owl dl entailments. In: Proc. of the 6th International Semantic Web Conference (ISWC-2007) and the 2nd Asian Semantic Web Conference (ASWC-2007). pp. 267–280. Springer (2007)
11. Lehmann, J.: DL-Learner: learning concepts in description logics. Journal of Machine Learning Research (JMLR) 10, 2639–2642 (2009)
12. Lehmann, J., Bühmann, L.: ORE – a tool for repairing and enriching knowledge bases. In: Proc. of the International Semantic Web Conference (ISWC), LNCS, vol. 6497, pp. 177–193. Springer Berlin / Heidelberg (2010)
13. Meilicke, C., Völker, J., Stuckenschmidt, H.: Learning disjointness for debugging mappings between lightweight ontologies. In: Knowledge Engineering: Practice and Patterns, LNCS, vol. 5268, pp. 93–108. Springer Berlin / Heidelberg (2008)
14. Qi, G., Haase, P., Huang, Z., Ji, Q., Pan, J.Z., Völker, J.: A kernel revision operator for terminologies - algorithms and evaluation. In: The Semantic Web – ISWC 2008. pp. 419–434. Springer Berlin / Heidelberg (2008)
15. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: International Joint Conference on Artificial Intelligence. vol. 18, pp. 355–362 (2003)
16. Sirin, E., Parsia, B., Grau, B.C., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics 5, 51–53 (2007)
17. Völker, J., Niepert, M.: Statistical schema induction. In: The Semantic Web: Research and Applications. LNCS, vol. 6643, pp. 124–138. Springer Berlin / Heidelberg (2011)
18. Völker, J., Rudolph, S.: Fostering web intelligence by semi-automatic OWL ontology refinement. In: Proc. of the 7th International Conference on Web Intelligence (WI). IEEE (December 2008)
19. Wu, G., Qi, G., Du, J.: Finding all justifications of owl entailments using tms and mapreduce. In: Proc. of the 20th ACM international conference on Information and knowledge management. pp. 1425–1434. ACM (2011)