ORE - A Tool for Repairing and Enriching Knowledge Bases

Jens Lehmann and Lorenz Bühmann

AKSW research group, University of Leipzig, Germany,
lastname@informatik.uni-leipzig.de

Abstract. While the number and size of Semantic Web knowledge bases increases, their maintenance and quality assurance are still difficult. In this article, we present ORE, a tool for repairing and enriching OWL ontologies. State-of-the-art methods in ontology debugging and supervised machine learning form the basis of ORE and are adapted or extended so as to work well in practice. ORE supports the detection of a variety of ontology modelling problems and guides the user through the process of resolving them. Furthermore, the tool allows to extend an ontology through (semi-)automatic supervised learning. A wizard-like process helps the user to resolve potential issues after axioms are added.

1 Introduction

Over the past years, the number and size of knowledge bases in the Semantic Web has increased significantly, which can be observed in various ontology repositories and the LOD cloud[1]. One of the remaining major challenges is, however, the maintenance of those knowledge bases and the use of expressive language features of the standard web ontology language OWL.

The goal of the ORE (Ontology Repair and Enrichment) tool[2] is to provide guidance for knowledge engineers who want to detect problems in their knowledge base and repair them. ORE also provides suggestions for extending a knowledge base by using supervised machine learning on the instance data in the knowledge base. ORE takes the web aspect of the Semantic Web into account by supporting large Web of Data knowledge bases like OpenCyc and DBpedia.

The main contributions of the article are as follows:

- provision of a free tool for repairing and extending ontologies
- implementation and combination of state-of-the-art inconsistency detection, ranking, and repair methods
- use of supervised learning for extending an ontology
- support for very large knowledge bases available as Linked Data or via SPARQL endpoints
- application tests of ORE on real ontologies

[1] http://linkeddata.org

[2] See http://dl-learner.org/wiki/ORE and download at http://sourceforge.net/projects/dl-learner/files/.

The article is structured as follows: In Section 2, we cover the necessary foundations in the involved research disciplines such as description logics (DLs), ontology debugging, and learning in OWL. Section 3 describes how ontology debugging methods were implemented and adapted in ORE. Similarly, Section 4 shows how an existing framework for ontology learning was incorporated. In Section 5, we describe the structure of the ORE user interface. The evaluation of both, the repair and enrichment part, is given in Section 6. Related work is presented in Section 7 followed by our final conclusions in Section 8.

2 Preliminaries

We give a brief introduction into DLs and OWL as the underlying formalism, recapitulate the state of the art in ontology debugging and give the definition of the class learning problem in ontologies.

2.1 Description Logics and OWL

DLs are usually decidable fragments of first order logic and have a variable-free syntax. The standard ontology language OWL 2 is based on the DL $\mathcal{SROIQ}$. We briefly introduce it and refer to [12] for details.

In $\mathcal{SROIQ}$, three sets are used as the base for modelling: *individual* names N_I, *concept* names N_C (called classes in OWL), and *role* names (object properties) N_R. By convention, we will use A, B (possibly with subscripts) for concept names, r for role names, a for individuals, and C, D for complex concepts. Using those basic sets, we can inductively build complex concepts using the following constructors:

$$A \mid \top \mid \bot \mid \{a\} \mid C \sqcap D \mid C \sqcup D$$
$$\mid \exists r.\texttt{Self} \mid \exists r.C \mid \forall r.C \mid \leq n\ r.c \mid \geq n\ r.C$$

For instance, $\texttt{Man} \sqcap \exists \texttt{hasChild}.\texttt{Female}$ is a complex concept describing a man who has a daughter. A *DL knowledge base* consists of a set of *axioms*. The signature of a knowledge base (an axiom α) is the set $\mathbf{S}$ (*Sig*(α)) of atomic concepts, atomic roles and individuals that occur in the knowledge base (in α). We will only mention two kinds of axioms explicitly: Axioms of the form $C \sqsubseteq D$ are called *general inclusion axioms*. An axiom of the form $C \equiv D$ is called *equivalence axiom*. In the special case that C is a concept name, we call the axiom a *definition*.

Apart from *explicit* knowledge, we can deduce *implicit* knowledge from a knowledge base. *Inference/reasoning algorithms* extract such implicit knowledge. Typical reasoning tasks are:

- instance check $\mathcal{K} \models C(a)$? (Does a belong to C?)
- retrieval $R_{\mathcal{K}}(C)$? (Determine all instances of C.)
- subsumption $C \sqsubseteq_{\mathcal{K}} D$? (Is C more specific than D?)
- inconsistency $\mathcal{K} \models$ false? (Does $\mathcal{K}$ contain contradictions?)
- satisfiability $C \equiv_{\mathcal{K}} \bot$? (Can C have an instance?)
- incoherence $\exists C\ (C \equiv_{\mathcal{K}} \bot)$? (Does K contain an unsatisfiable class?)

Throughout the paper, we use the words ontology and knowledge base as well as complex concept and class expression synonymously.

2.2 Ontology Debugging

Finding and understanding undesired entailments such as unsatisfiable classes or inconsistency can be a difficult or impossible task without tool support. Even in ontologies with a small number of logical axioms, there can be several, non-trivial causes for an entailment. Therefore, interest in finding explanations for such entailments has increased in recent years. One of the most usual kinds of explanations are *justifications* [15]. A justification for an entailment is a minimal subset of axioms with respect to a given ontology, that is sufficient for the entailment to hold. More formally, let $\mathcal{O}$ be a given ontology with $\mathcal{O} \models \eta$, then $\mathcal{J}$ is a justification for η if $\mathcal{J} \models \eta$, and for all $\mathcal{J}' \subset \mathcal{J}$, $\mathcal{J}' \not\models \eta$. In the meantime, there is support for the detection of potentially overlapping justifications in tools like Protégé[3] and Swoop[4]. Justifications allow the user to focus on a small subset of the ontology for fixing a problem. However, even such a subset can be complex, which has spurred interest in computing *fine-grained* justifications [11] (in contrast to *regular* justifications). In particular, *laconic justifications* are those where the axioms do not contain superfluous parts and are as weak as possible. A subset of laconic justifications are *precise justifications*, which split larger axioms into several smaller axioms allowing minimally invasive repair.

A possible approach to increase the efficiency of computing justifications is module extraction [6]. Let $\mathcal{O}$ be an ontology and $\mathcal{O}' \subseteq \mathcal{O}$ a subset of axioms of $\mathcal{O}$. $\mathcal{O}'$ is a module for an axiom α with respect to $\mathcal{O}$ if: $\mathcal{O}' \models \alpha$ iff $\mathcal{O} \models \alpha$. $\mathcal{O}'$ is a module for a signature $\mathbf{S}$ if for every axiom α with $\mathit{Sig}(\alpha) \subseteq \mathbf{S}$, we have that $\mathcal{O}'$ is a module for α with respect to $\mathcal{O}$. Intuitively, a module is an ontology fragment, which contains all relevant information in the ontology with respect to a given signature. One possibility to extract such a module is syntactic locality [6]. [30] showed that such *locality-based modules* contain all justifications with respect to an entailment and can provide order-of-magnitude performance improvements.

2.3 The Class Learning Problem

The process of learning in logics, i.e. trying to find high level explanations for given data, is also called *inductive reasoning* as opposed to the deductive reasoning tasks we have introduced. The main difference is that in deductive reasoning it is formally shown whether a statement follows from a knowledge base, whereas in inductive learning we invent new statements. Learning problems, which are similar to the one we will analyse, have been investigated in *Inductive Logic Programming* [27] and, in fact, the method presented here can be used to solve a variety of machine learning tasks apart from ontology engineering.

The considered supervised ontology learning problem is an adaption of the problem in *Inductive Logic Programming*. We learn a formal description of a class A from inferred instances in the ontology. Let a class name $A \in N_C$ and an ontology $\mathcal{O}$ be given. We define the *class learning problem* as finding a class expression C such that $R_{\mathcal{O}}(C) = R_{\mathcal{O}}(A)$, i.e. C covers exactly all instances of A.

[3] `http://protege.stanford.edu`
[4] `http://www.mindswap.org/2004/SWOOP/`

Clearly, the learned concept C is a description of (the instances of) A. Such a concept is a candidate for adding an axiom of the form $A \equiv C$ or $A \sqsubseteq C$ to the knowledge base $\mathcal{K}$. This is used in the enrichment step in ORE as we will later describe. In the case that A is described already via axioms of the form $A \sqsubseteq C$ or $A \equiv C$, those can be either modified, i.e. specialised/generalised, or relearned from scratch by learning algorithms.

Machine learning algorithms usually prefer those solutions of a learning problem, which are likely to classify unknown individuals well. For instance, using nominals (`owl:oneOf`) to define the class A above as the set of its current instances is a correct solution of the learning problem, but would classify all individuals, which are added to the knowledge base later as not being instance of A. In many cases, the learning problem is not perfectly solvable apart from the trivial solution using nominals. In this case, approximations can be given by ML algorithms. It is important to note that a knowledge engineer usually makes the final decision on whether to add one of the suggested axioms, i.e. candidate concepts are presented to the knowledge engineer, who can then select and possibly refine one of them.

3 Ontology Repair

For a single entailment, e.g. an unsatisfiable class, there can be many justifications. Moreover, in real ontologies, there can be several unsatisfiable classes or several reasons for inconsistency. While the approach described in Section 2.2 works well for small ontologies, it is not feasible if a high number of justifications or large justifications have to be computed. Due to the relations between entities in an ontology, several problems can be intertwined and are difficult to separate. We briefly describe how we handle these problems in ORE.

Root Unsatisfiability For the latter problem mentioned above, an approach [18] is to separate between root and derived unsatisfiable classes. A derived unsatisfiable class has a justification, which is a proper super set of a justification of another unsatisfiable class. Intuitively, their unsatisfiability may depend on other unsatisfiable classes in the ontology, so it can be beneficial to fix those root problems first. There are two different approaches for determining such classes: The first approach is to compute all justifications for each unsatisfiable class and then apply the definition. The second approach relies on a structural analysis of axioms and heuristics. Since the first approach is computationally too expensive for larger ontologies, we use the second strategy as default in ORE. The implemented approach is sound, but incomplete, i.e. not all class dependencies are found, but the found ones are correct. To increase the proportion of found dependencies, the TBox is modified in a way which preserves the subsumption hierarchy to a large extent. It was shown in [18] that this allows to draw further entailments and improve the pure syntactical analysis.

Axiom Relevance Given a justification, the problem needs to be resolved by the user, which involves the deletion or modification of axioms in it. To assist the user, ranking methods, which highlight the most probable causes for problems, are important. Common methods (see [16] for details) are frequency (How often does the axiom appear in justifications?), syntactic relevance (How deeply rooted is an axiom in the ontology?)

and semantic relevance (How many entailments are lost or added?[5]). ORE supports all metrics and a weighted aggregation of them. For computing semantic relevance, ORE uses the incremental classification feature of Pellet, which uses locality-based modules. Therefore, only the relevant parts of the ontology are reclassified when determining the effect of changes.

Consequences of Repair Step Repairing a problem involves editing or deleting an axiom. Deletion has the technical advantage that it does not lead to further entailments due to the monotonicity of DLs. However, desired entailments may be lost. In contrast, editing axioms allows to make small changes, but it may lead to new entailments, including inconsistencies. To support the user, ORE provides fine-grained justifications, which only contain relevant parts of axioms and, therefore, have minimal impact on deletion. Furthermore, ORE allows to preview new or lost entailments. The user can then decide to preserve them, if desired.

Workflow The general workflow of the ontology repair process is depicted in Figure 1. First, all inconsistencies are resolved. Secondly, unsatisfiable classes are handled by computing root unsatisfiable classes, as well as regular and laconic justifications, and different ranking metrics.

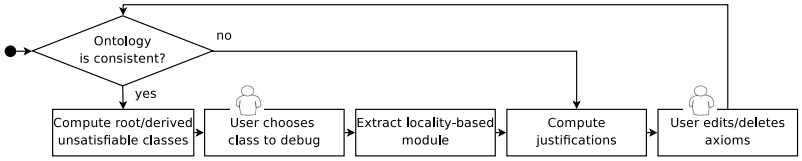

Fig. 1. Workflow for debugging an ontology in ORE.

Web of Data and Scalability In order to apply ORE to existing very large knowledge bases in the Web of Data, the tool supports using SPARQL endpoints instead of local OWL files as input knowledge bases. To perform reasoning on those knowledge bases, ORE implements an incremental load procedure inspired by [9].

Using SPARQL queries, the knowledge base is loaded in small chunks. In the first step, ORE determines the size of the knowledge base by determining the number of all types of OWL 2 axioms. In the main part of the algorithm, a priority based loading procedure is used. This means that axioms that are empirically more likely to cause inconsistencies in the sense that they are often part of justifications have a higher priority. In general, schema axioms have a higher loading priority than instance data. Before loading parts of the instance data, the algorithm performs sanity checks on the data, i.e. performs a set of simple SPARQL queries, which probe for inconsistent axiom sets. These cases include individuals, which are instances of disjoint classes, properties which are used on instances incompatible with their domain, etc. The algorithm can also be configured to fetch additional information via Linked Data such that consistency of

[5] Since the number of entailed axioms can be infinite, we restrict ourselves to a subset of axioms as suggested in [16].

a knowledge base in combination with knowledge from another knowledge base can be tested.

The algorithm converges towards loading the whole knowledge base into the reasoner, but can also be configured to stop automatically after the schema part and sample instances, based on ABox summarisation techniques, of all classes have been loaded. This is done to prevent a too high load on SPARQL endpoints and the fact that most knowledge bases cannot be loaded into standard OWL reasoners on typical hardware available. At the moment, the algorithm uses the incremental reasoning feature available in Pellet such that it is not required to reload the reasoner each time a chunk of data has been received from the SPARQL endpoint.

The general idea behind this component of ORE is to apply state-of-the-art reasoning methods on a larger scale than was possible previously. We show this by applying ORE on OpenCyc and DBpedia in Section 6.3. To the best of our knowledge, none of the existing tools can compute justifications for inconsistencies on those large knowledge bases. This part of ORE aims at stronger support for the "web aspect" of the Semantic Web and the high popularity of Web of Data initiative.

4 Ontology Enrichment

Currently, ORE supports enriching an ontology with axioms of the form $A \equiv C$ and $A \sqsubseteq C$. For suggesting such an axiom, we use the DL-Learner framework to solve the class learning problem described in Section 2.3. In particular, we use the CELOE algorithm in DL-Learner, which is optimised for class learning in an ontology engineering scenario. It is a specialisation of the OCEL algorithm [24], which was shown to be very competitive.

The main task of ORE is to provide an interface to the algorithm and handle the consequences of adding a suggested axiom. In this section, we will focus on the latter problem. The learning algorithm can produce false positives as well as false negatives, which can lead to different consequences. In the following, assume $\mathcal{O}$ to be an ontology and A the class for which a definition $A \equiv C$ was learned. Let n be a false positive, i.e. $\mathcal{O} \not\models A(n)$ and $\mathcal{O} \models C(n)$. We denote the set of justifications for $\mathcal{O} \models \eta$ with $\mathcal{J}_\eta$. ORE would offer the following options in this case:

1. assign n to class A
2. completely delete n in $\mathcal{O}$
3. modify assertions about n such that $\mathcal{O} \not\models C(n)$: In a first step, ORE uses several reasoner requests to determine the part C' of C, which is responsible for classifying n as instance of C. The algorithm recursively traverses conjunctions and disjunctions until it detects one of the class constructors below.
   - $C' = B$ $(B \in N_C)$: Remove the assignment of n to B, i.e. delete at least one axiom in each justification $J \in \mathcal{J}_{B(n)}$
   - $C' = \forall r.D$: Add at least one axiom of the form $r(n, a)$ where a is not an instance of D
   - $C' = \exists r.D$:
     (a) Remove all axioms of the form $r(n, a)$, where a is an instance of D

(b) Remove all axioms of the form $r\,(n, a)$

- $C' =\leq mr.D$: Add axioms of the form $r\,(n, a)$, in which a is instance of D, until their number is greater than m
- $C' =\geq mr.D$: Remove axioms of the form $r\,(n, a)$, where a is instance of D, until their number is smaller than m

The steps above are an excerpt of the provided functionality of ORE. False negatives are treated in a similar fashion. The strategy is adapted in case of learning superclass axioms ($A \sqsubseteq C$). Those steps, where axioms are added, can naturally lead to inconsistencies. In such a case, a warning is displayed. If the user chooses to execute the action, the ORE wizard can return to the inconsistency resolution step described in Section 2.2.

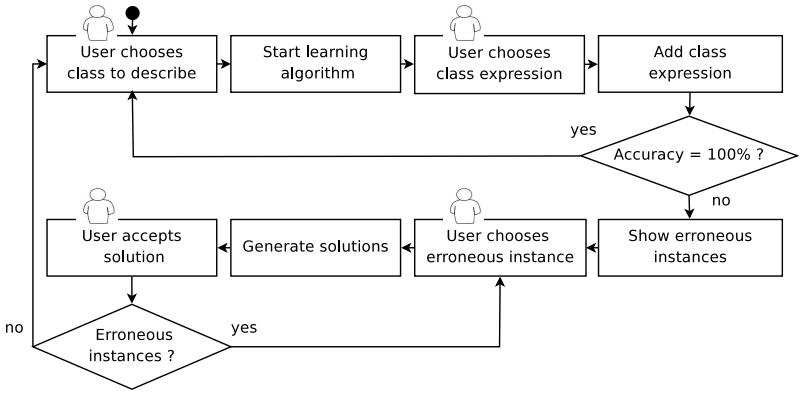

Fig. 2. Workflow for enriching an ontology in ORE.

Workflow The workflow for the enrichment process is shown in Figure 2. First, the user selects a class for which he wants to learn a description. Alternatively, ORE can loop over all classes and provides particularly interesting suggestions to the user. ORE calls the CELOE learning algorithm and presents the 10 best suggestions to the user. If the user decides to accept a suggestion and if there are false positives or negatives, possible repair solutions are provided.

5 User Interface

In ORE, we decided to use a wizard-based user interface approach. This allows a user to navigate through the dialogues step-by-step, while the dependencies between different steps are factored in automatically. This enables the user to perform the repair and enrichment process with only a few clicks and a low learning curve. Changes can be rolled back if necessary. The design of ORE ensures that it can be embedded in ontology editors. Below, we describe the most important parts of the ORE wizard.

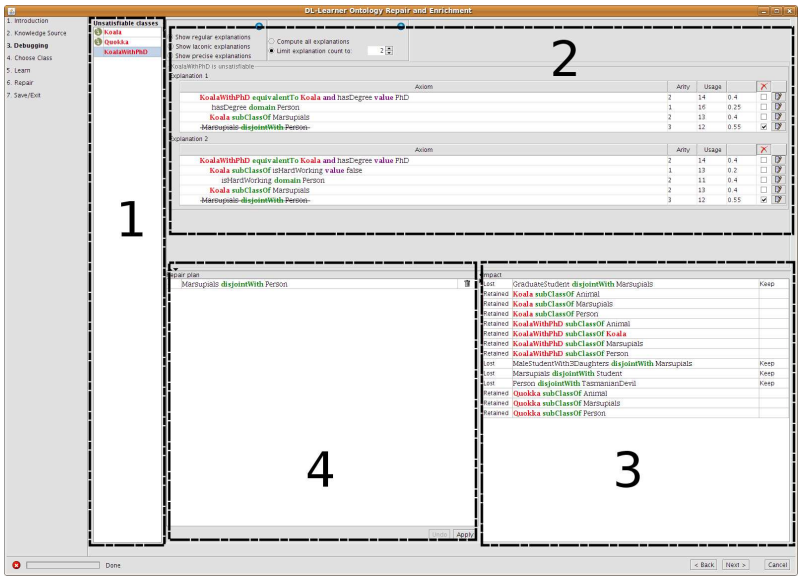

Fig. 3. The panel for debugging the ontology.

Debugging Phase The debugging panel is separated into four parts (see Figure 3): The left part (1) contains unsatisfiable classes for the case that the considered ontology is consistent. Unsatisfiable root classes are marked with a symbol in front of their name. The upper part (2) shows the computed justifications. In addition to listing the axioms, several metrics are displayed in a table as well as the actions for removing or editing the axiom. The axioms are displayed in Manchester OWL Syntax[6]. To increase readability, key words are emphasised and the axioms are indented. Configuration options allow to set the maximum number of explanations, which should be displayed, and their type (regular/laconic). In (3), lost or added entailments, as a consequence of the selected modifications, are displayed. This part of the user interface allows to preserve those entailments, if desired. Part (4) of the debugging panel lists the axioms, which will be added or removed. Each action can be undone. When a user is satisfied with the changes made, they can execute the created repair plan, which results in the actual modification of the underlying ontology.

Enrichment Phase For the enrichment phase, the panel is separated into three parts (see Figure 4). The right part (2) allows to start or stop the underlying machine learning algorithm, the configuration of it, and the selection whether equivalent or superclass axioms should be learned. In part (1), the learned expressions are displayed. For each class expression, a heuristic accuracy value provided by the underlying algorithm is displayed. When a class expression is selected, an illustration of its coverage is shown in part (3). The illustration is generated by analysing the instances covered by the class expression and comparing it to the instance of the current named class.

[6] `http://www.w3.org/2007/OWL/wiki/ManchesterSyntax`

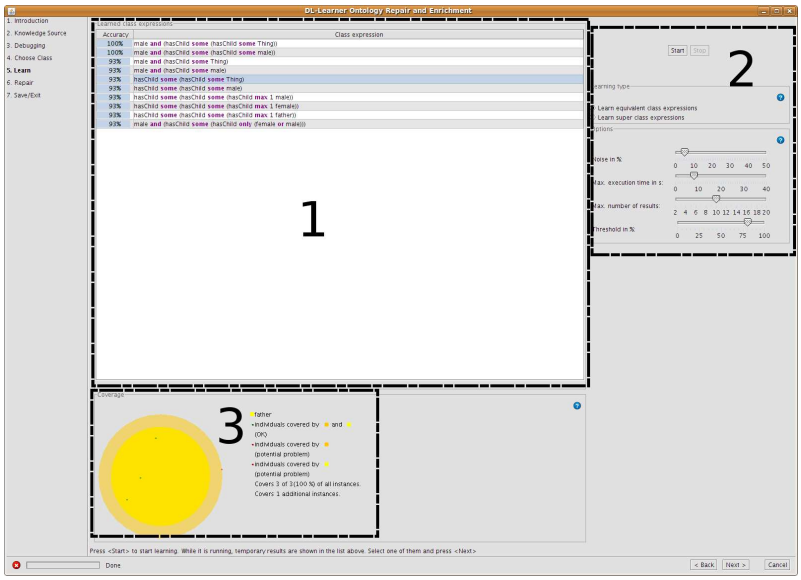

Fig. 4. The panel for enriching the ontology.

Repair of Individuals Enriching the ontology can have consequences on the classification of individuals in the ontology. For repairing unwanted consequences, a dialogue (see Figure 5) is displayed, which is separated in three parts. The upper part (1) shows the class expression itself. As briefly described in Section 4, the parts of the expression, which cause the problem, are highlighted. Clicking on such a part of an expression, opens a menu, which provides repair suggestions. The middle part (2) displays information about the individual, which is currently repaired. This allows the ontology engineer to observe relevant information at a glance. The lower part lists the repair decisions made and provides an undo method.

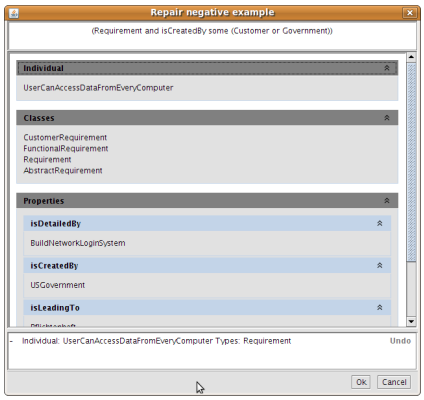

Fig. 5. The panel for repairing an erroneous instance.

6 Application to Existing Knowledge Bases

To test the ORE tool, we used the TONES and Protégé ontology repositories. We loaded all ontologies in those repositories into the Pellet reasoner. Inconsistent and incoherent ontologies were selected as evaluation candidates for the repair step and all ontologies which contain at least 5 classes with at least 3 instances, were selected as candidates for the enrichment step. Out of 216 ontologies which could be loaded into the reasoner, 3 were inconsistent, and 32 were incoherent.

Please note that we have not performed an extensive evaluation of all methods underlying ORE as this has been done in the cited articles, where the methods are described in more detail. The main objective was to find out whether the tool is applicable to real-world ontologies with respect to usability, performance, and stability.

6.1 Repair Step

This part of our tests was performed by the authors of the article. From the 35 candidate ontologies, we selected 7 ontologies where we could obtain an understanding of the domain within one working day. These ontologies and the test results are shown in Table 1. We used ORE to resolve all occurring problems and, overall, resolved 1 inconsistency and 135 unsatisfiable classes. Generally, the ontologies could be processed without problems and the performance for computing justifications was sufficient. The maximum time required per justification was one second.

ontology	resolved inconsistency	#resolved unsatisfiable classes	#removed axioms	#added axioms	#changed axioms
http://protege.cim3.net/file/pub/ontologies/camera/**camera.owl**	yes	-	0	0	2
http://protege.cim3.net/file/pub/ontologies/koala/**koala.owl**	-	3	3	0	0
http://reliant.teknowledge.com/DAML/**Economy.owl**	-	51	11	5	0
http://www.cs.man.ac.uk/ horridgm/ontologies/complexity/**UnsatCook.owl**	-	8	1	0	0
http://www.co-ode.org/ontologies/pizza/2007/02/12/**pizza.owl**	-	2	3	0	0
http://www.mindswap.org/ontologies/debugging/**University.owl**	-	9	3	1	2
http://reliant.teknowledge.com/DAML/**Transportation.owl**	-	62	15	28	0

Table 1. Repair of ontologies from the Protégé and TONES repositories.

6.2 Enrichment Step

The test of the enrichment step was done by two researchers, who made themselves familiar with the domain of the test ontologies. We are aware that an ideal evaluation procedure would require OWL knowledge engineers from the respective domains, e.g. different areas within biology, medicine, finance, and geography. Considering the budget limitations, however, we believe that our method is sufficient to be able to meet our basic test objectives for the first releases of ORE. Each researcher worked independently and had to make 383 decisions, as described below. The time required to make those decisions was 40 working hours per researcher.

ontology	#logical axioms	#suggestion lists	accept (1) in %	reject (2) in %	fail (3) in %	selected position on suggestion list (incl. std. deviation)	#hidden inconsistencies	#additional instances
http://www.mindswap.org/ontologies/**SC.owl**	20081	12	79	21	0	2.2±2.1	0	1771
http://www.fadyart.com/ontologies/data/**Finance.owl**	16057	50	52	48	0	3.6±2.6	0	1162
http://www.biopax.org/release/**biopax-level2.owl**	12381	34	78	22	0	2.7±2.2	1	803
http://i2geo.net/ontologies/dev/**GeoSkills.owl**	8803	180	56	44	0	1.6±1.2	1	295
http://reliant.teknowledge.com/DAML/**Economy.owl**	1625	22	74	26	0	1.5±0.9	0	77
http://www.acl.icnet.uk/ mw/**MDM0.73.owl**	884	77	56	44	0	3.7±2.6	1	82
http://www.co-ode.org/ontologies/.../**eukariotic.owl**	38	8	91	9	0	2.5±1.2	0	7

Table 2. Test results on several ontologies. On average, suggestions by the ML algorithm were acepted in 60% of all cases.

From those ontologies obtained in the pre-selection step, described at the beginning of this section, we picked ontologies, which vary in size and complexity. We wanted to determine whether 1.) the underlying adapted learning algorithm is useful in practice, i.e. is able to make sensible suggestions, 2.) to which extent additional information can be inferred when enriching ontologies with suggestions by the learning algorithm (described as *hidden inconsistencies* and *additional instances* below).

We ran ORE in an evaluation mode, which works as follows: For each class A, the learning method generates at most ten suggestions with the best ones on top of the list. This is done for learning superclasses ($A \sqsubseteq C$) and equivalent classes ($A \equiv C$) separately. If the accuracy of the best suggestion exceeds a defined threshold, we suggest them to the knowledge engineer. The knowledge engineer then has three options to choose from: 1. pick one of the suggestions by entering its number (accept), 2. declare that there is no sensible suggestion for A in his opinion (reject), or 3. declare that there is a sensible suggestion, but the algorithm failed to find it (fail). If the knowledge engineer decides to pick a suggestion, we query whether adding it leads to an inconsistent ontology. We call this case the discovery of a *hidden inconsistency*, since it was present before, but can now be formally detected and treated. We also measure whether adding the suggestion increases the number of inferred instances of A. Being able to infer *additional instances* of A, therefore, provides added value (see also the notion of *induction rate* as defined in [5]).

We used the default settings of 5% noise and an execution time of 10 seconds for the algorithm. The evaluation machine was a notebook with a 2 GHz CPU and 3 GB RAM. Table 2 shows the evaluation results.

Objective 1: We can observe that the researchers picked option 1 (accept) most of the time, i.e. in many cases the algorithm provided meaningful suggestions. This allows us to answer the first evaluation objective positively. The researchers never declared that the algorithm failed on finding a potential solution. The 7th column shows that many selected expressions are amongst the top 5 (out of 10) in the suggestion list, i.e. providing 10 suggestions appears to be a reasonable choice.

Objective 2: In 3 cases a hidden inconsistency was detected. Both researchers independently coincided on those decisions. The last column shows that in all ontologies additional instances could be inferred for the classes to describe if the new axiom would be added to the ontology after the learning process. Overall, being able to infer additional instances was very common and hidden inconsistencies could sometimes be detected.

6.3 Very Large Knowledge Bases

We applied ORE to two very large knowledge bases: DBpedia [21] (live version [10]) and OpenCyc. DBpedia is a knowledge base extracted from Wikipedia in a joint effort of the University of Leipzig, the Free University of Berlin and the company OpenLink. It contains descriptions of over 3.4 million entities out of which 1.5 million are classified in the DBpedia ontology. Overall, the DBpedia knowledge base consists of more than one billion triples with more than 250 million triples in the English language edition. OpenCyc is a part of the Cyc artificial intelligence project started in 1984, which provides a huge knowledge base of common sense knowledge. In its current OWL version, it contains more than 50 thousand classes, 20 thousand properties, 350 thousand invididuals. OpenCyc has a sophisticated and large schema, while DBpedia has a smaller schema part and a huge amount of instance data.

Application to DBpedia Most reasoning on DBpedia focuses on very light-weight reasoning techniques, which are usually employed within triple stores like OpenLink Virtuoso. Standard OWL reasoners are not able to load or reason within DBpedia. However, the incremental approach sketched in Section 3 allows to apply standard reasoners to DBpedia, detect inconsistencies and compute justifications with only moderate hardware requirements. Two justifications in Manchester Syntax are shown below[7]:

Example 1 (Incorrect Property Range in DBpedia).
Individual: dbr:Purify_%28album%29 Facts: dbo:artist dbr:Axis_of_Advance
Individual: dbr:Axis_of_Advance Types: dbo:Organisation
Class: dbo:Organisation DisjointWith dbo:Person
ObjectProperty: dbo:artist Range: dbo:Person

ORE found an assertion that "Axis of Advance" created the album "Purify". DBpedia states that the range of the "artist" property is a person, hence "Axis of Advance" must be a person. However, it is an organisation (a music band) and organisations and persons are disjoint, so we get a contradiction. In this example, the problem can be resolved by generalising the range of "artist", i.e. not requiring it to be a person.

Example 2 (DBpedia Incompatible with External Ontology).
Individual: dbr:WKWS Facts: geo:long -81.76833343505859
Types: dbo:Organisation
DataProperty: geo:long Domain: geo:SpatialThing
Class: dbo:Organisation DisjointWith: geo:SpatialThing

[7] Used prefixes: dbr = http://dbpedia.org/resource/, dbo = http://dbpedia.org/ontology/, geo = http://www.w3.org/2003/01/geo/wgs84_pos#

In this example, the longitude property is used on an organisation, which is a contradiction, because an organisation is itself not a spatial entity. The interesting aspect of this example is that information from an external knowledge base (W3C Geo) is fetched via Linked Data, which is an optional feature of ORE. The inconsistency only arises in combination with this external knowledge.

Application to OpenCyc OpenCyc is very large, but still loadable in standard OWL reasoners. However, only few reasoners can detect that it is not consistent. In our experiments, only HermiT 1.2.3 was able to do this given sufficient memory. Nevertheless, computing actual justifications is still not possible when considering the whole knowledge base and could only be achieved using the incremental priority-based load procedure in ORE. Below is an inconsistency detected by ORE in "label view", i.e. the value of `rdfs:label` is shown instead of the URIs:

Example 3 (Class Hierarchy Problems in OpenCyc).
Individual: 'PopulatedPlace' Types: 'ArtifactualFeatureType', 'ExistingStuffType'
Class: 'ExistingObjectType' DisjointWith: 'ExistingStuffType'
Class: 'ArtifactualFeatureType' SubClassOf: 'ExistingObjectType'

The example shows a problem in OpenCyc, where an individual is assigned to classes, which can be inferred to be disjoint via the class hierarchy. (Note that "PopulatedPlace" is used as individual and class in OpenCyc, which is allowed in OWL2.)

7 Related Work

The growing interest in Semantic Technologies has led to an increasing number of ontologies, which has, in turn, spurred interest in techniques for ontology creation and maintenance. In [28] and [8], methods for the detection and repair of inconsistencies in frequently changing ontologies were developed. [29] discusses a method for axiom pinpointing, i.e. the detection of axioms responsible for logical errors. A non proof-theoretic method is used in OntoClean [7]. By adding meta-properties (rigidity, identity, dependency) to each class, problems in the knowledge base taxonomy could be identified by using rules. Classes could then be moved in the hierarchy or additional ones can be added. OntoClean supports resolving taxonomy errors, but was not designed for detecting logical errors.

The work on the enrichment part of ORE goes back to early work on supervised learning in DLs, e.g. [3], which used so-called least common subsumers to solve the learning problem (a modified variant of the problem defined in this article). Later, [2] invented a refinement operator for $\mathcal{ALER}$ and proposed to solve the problem by using a top-down approach. [4,13] combine both techniques and implement them in the YINYANG tool. However, those algorithms tend to produce very long and hard-to-understand class expressions, which are often not appropriate in an ontology enrichment context. Therefore, ORE is based on DL-Learner [20], which allows to select between a variety of learning algorithms [22,23,19,24]. Amongst them, CELOE is particularly optimised for learning easy to understand expressions. DL-FOIL [5] is a similar approach mixing upward and downward refinement. Other approaches focus on learning in hybrid language settings [26].

In (semi-)automatic ontology engineering, formal concept analysis [1] and relational exploration [32] have been used for completing knowledge bases. [33] focuses on learning disjointness between classes in an ontology to allow for more powerful reasoning and consistency checking. Naturally, there is also a lot of research work on ontology learning from text. The most closely related approach in this area is [31], in which OWL DL axioms are obtained by analysing sentences which have definitorial character.

There are a number of related tools for ontology repair:

Swoop[8]**[17]** is a Java-based ontology editor using web browser concepts. It can compute justifications for the unsatisfiability of classes and offers a repair mode. The fine-grained justification computation algorithm is, however, incomplete. Swoop can also compute justifications for an inconsistent ontology, but does not offer a repair mode like ORE in this case. It does not extract locality-based modules, which leads to lower performance for large ontologies.

RaDON[9]**[14]** is a plugin for the NeOn toolkit. It offers a number of techniques for working with inconsistent or incoherent ontologies. It can compute justifications and, similarly to Swoop, offers a repair mode. RaDON also allows to reason with inconsistent ontologies and can handle sets of ontologies (ontology networks). Compared to ORE, there is no feature to compute fine-grained justifications, and the user gets no informations about the impact of repair.

Pellint[10]**[25]** is a Lint-based tool, which searches for common patterns which lead to potential reasoning performance problems. In future work, we plan to integrate support for detecting and repairing reasoning performance problems in ORE.

PION and DION[11] have been developed in the SEKT project to deal with inconsistencies. PION is an inconsistency tolerant reasoner, i.e. it can, unlike standard reasoners, return meaningful query answers in inconsistent ontologies. To achieve this, a four-valued paraconsistent logic is used. DION offers the possibility to compute justifications, but cannot repair inconsistent or incoherent ontologies.

Explanation Workbench[12] is a Protégé plugin for reasoner requests like class unsatisfiability or inferred subsumption relations. It can compute regular and laconic justifications [11], which contain only those axioms which are relevant for answering the particular reasoner request. This allows to make minimal changes to resolve potential problems. We adapted its layout for the ORE debugging interface. Unlike ORE, the current version of Explanation Workbench does not allow to remove axioms in laconic justifications.

Most of those tools were designed to detect logical errors or ignore them (PION). PellInt is an exception because it detects problems relevant for reasoning performance. The ORE tool unites several techniques present in those tools and combines them with

[8] SWOOP: `http://www.mindswap.org/2004/SWOOP/`
[9] RaDON: `http://radon.ontoware.org/demo-codr.htm`
[10] PellInt: `http://pellet.owldl.com/pellint`
[11] PION: `http://wasp.cs.vu.nl/sekt/pion/`
DION: `http://wasp.cs.vu.nl/sekt/dion/`
[12] `http://owl.cs.manchester.ac.uk/explanation/`

the DL-Learner framework to enable suggestions for enrichment. It also enhances other tools by providing support for working on SPARQL endpoints and Linked Data.

8 Conclusions and Future Work

We have presented a freely available tool for ontology repair and enrichment. It integrates state-of-the-art methods from ontology debugging and supervised learning in OWL in an intuitive, wizard-like interface. It combines the advantages of other tools and provides new functionality like the enrichment part of the tool. An evaluation on real ontologies has shown the need for a repair and enrichment tool and, in particular, the benefits of ORE.

In future work, we aim at integrating support for further modelling problems apart from inconsistencies and unsatisfiable classes. Those problems will be ordered by severity reaching from logical problems to suggested changes for improving reasoner performance. We plan to improve the enrichment part by suggesting other types of axioms, e.g. disjointness. We also plan to evaluate and optimise the SPARQL/Linked Data component of ORE. Possibly, we will provide an alternative web interface and appropriate hardware infrastructure for ORE such that it can be used for online analysis of Web of Data knowledge bases. In addition to those features, a constant evaluation of the underlying methods will be performed to improve the foundations of ORE.

References

1. Franz Baader, Bernhard Ganter, Ulrike Sattler, and Baris Sertkaya. Completing description logic knowledge bases using formal concept analysis. In *IJCAI 2007*. AAAI Press, 2007.
2. Liviu Badea and Shan-Hwei Nienhuys-Cheng. A refinement operator for description logics. In *ILP 2000*, volume 1866 of *LNAI*, pages 40–59. Springer, 2000.
3. William W. Cohen and Haym Hirsh. Learning the CLASSIC description logic: Theoretical and experimental results. In *KR 94*, pages 121–133. Morgan Kaufmann, 1994.
4. Floriana Esposito, Nicola Fanizzi, Luigi Iannone, Ignazio Palmisano, and Giovanni Semeraro. Knowledge-intensive induction of terminologies from metadata. In *ISWC 2004*, volume 3298 of *LNCS*, pages 441–455. Springer, 2004.
5. Nicola Fanizzi, Claudia d'Amato, and Floriana Esposito. DL-FOIL concept learning in description logics. In *ILP 2008*, volume 5194 of *LNCS*, pages 107–121. Springer, 2008.
6. Bernardo Cuenca Grau, Ian Horrocks, Yevgeny Kazakov, and Ulrike Sattler. Modular reuse of ontologies: Theory and practice. *J. Artif. Intell. Res. (JAIR)*, 31:273–318, 2008.
7. Nicola Guarino and Christopher A. Welty. An overview of ontoclean. In Steffen Staab and Rudi Studer, editors, *Handbook on Ontologies*, pages 151–172. Springer, 2004.
8. Peter Haase, Frank van Harmelen, Zhisheng Huang, Heiner Stuckenschmidt, and York Sure. A framework for handling inconsistency in changing ontologies. In *ISWC 2005*, volume 3729 of *LNCS*, pages 353–367, Galway, Ireland, 2005. Springer.
9. Sebastian Hellmann, Jens Lehmann, and Sören Auer. Learning of OWL class descriptions on very large knowledge bases. *Int. Journal on Semantic Web and Information Systems*, 5(2):25–48, 2009.
10. Sebastian Hellmann, Claus Stadler, Jens Lehmann, and Sören Auer. Dbpedia live extraction. In *Proc. of 8th International Conference on Ontologies, DataBases, and Applications of Semantics (ODBASE)*, volume 5871 of *Lecture Notes in Computer Science*, pages 1209–1223, 2009.
11. Matthew Horridge, Bijan Parsia, and Ulrike Sattler. Laconic and precise justifications in OWL. In *The Semantic Web - ISWC 2008*, volume 5318 of *LNCS*, pages 323–338. Springer, 2008.

12. Ian Horrocks, Oliver Kutz, and Ulrike Sattler. The even more irresistible SROIQ. In *KR 2006*, pages 57–67. AAAI Press, 2006.
13. Luigi Iannone, Ignazio Palmisano, and Nicola Fanizzi. An algorithm based on counterfactuals for concept learning in the semantic web. *Applied Intelligence*, 26(2):139–159, 2007.
14. Qiu Ji, Peter Haase, Guilin Qi, Pascal Hitzler, and Steffen Stadtmüller. Radon - repair and diagnosis in ontology networks. In *ESWC 2009*, volume 5554 of *LNCS*, pages 863–867. Springer, 2009.
15. Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In *ISWC 2007*, volume 4825 of *LNCS*, pages 267–280, Berlin, Heidelberg, 2007. Springer.
16. Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and Bernardo Cuenca Grau. Repairing unsatisfiable concepts in owl ontologies. In *ESWC 2006*, volume 4011 of *LNCS*, pages 170–184, 2006.
17. Aditya Kalyanpur, Bijan Parsia, Evren Sirin, Bernardo Cuenca Grau, and James Hendler. Swoop: A web ontology editing browser. *Journal of Web Semantics*, 4(2):144–153, 2006.
18. Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. *Journal of Web Semantics*, 3(4):268–293, 2005.
19. Jens Lehmann. Hybrid learning of ontology classes. In *MLDM 2007*, volume 4571 of *LNCS*, pages 883–898. Springer, 2007.
20. Jens Lehmann. DL-Learner: learning concepts in description logics. *Journal of Machine Learning Research (JMLR)*, 10:2639–2642, 2009.
21. Jens Lehmann, Chris Bizer, Georgi Kobilarov, Sören Auer, Christian Becker, Richard Cyganiak, and Sebastian Hellmann. DBpedia - a crystallization point for the web of data. *Journal of Web Semantics*, 7(3):154–165, 2009.
22. Jens Lehmann and Pascal Hitzler. Foundations of refinement operators for description logics. In *ILP 2007*, volume 4894 of *LNCS*, pages 161–174. Springer, 2008.
23. Jens Lehmann and Pascal Hitzler. A refinement operator based learning algorithm for the ALC description logic. In *ILP 2007*, volume 4894 of *LNCS*, pages 147–160. Springer, 2008.
24. Jens Lehmann and Pascal Hitzler. Concept learning in description logics using refinement operators. *Machine Learning journal*, 78(1-2):203–250, 2010.
25. Harris Lin and Evren Sirin. Pellint - a performance lint tool for pellet. In *OWLED 2008*, volume 432 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2008.
26. Francesca A. Lisi. Building rules on top of ontologies for the semantic web with inductive logic programming. *TPLP*, 8(3):271–300, 2008.
27. Shan-Hwei Nienhuys-Cheng and Ronald de Wolf, editors. *Foundations of Inductive Logic Programming*, volume 1228 of *LNAI*. Springer, 1997.
28. Peter Plessers and Olga De Troyer. Resolving inconsistencies in evolving ontologies. In *ESWC 2006*, volume 4011 of *LNCS*, pages 200–214. Springer, 2006.
29. Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *IJCAI 2003*, pages 355–362. Morgan Kaufmann, 2003.
30. Boontawee Suntisrivaraporn, Guilin Qi, Qiu Ji, and Peter Haase. A modularization-based approach to finding all justifications for OWL DL entailments. In *ASWC 2008*, volume 5367 of *LNCS*, pages 1–15. Springer, 2008.
31. Johanna Völker, Pascal Hitzler, and Philipp Cimiano. Acquisition of OWL DL axioms from lexical resources. In *ESWC 2007*, volume 4519 of *LNCS*, pages 670–685. Springer, 2007.
32. Johanna Völker and Sebastian Rudolph. Fostering web intelligence by semi-automatic OWL ontology refinement. In *Web Intelligence*, pages 454–460. IEEE, 2008.
33. Johanna Völker, Denny Vrandecic, York Sure, and Andreas Hotho. Learning disjointness. In *ESWC 2007*, volume 4519 of *LNCS*, pages 175–189. Springer, 2007.