Get my pizza right: Repairing missing is-a relations in ALC ontologies

Patrick Lambrix, Zlatan Dragisic and Valentina Ivanova

Linköping University Post Print

N.B.: When citing this work, cite the original article.

Original Publication:

Patrick Lambrix, Zlatan Dragisic and Valentina Ivanova, Get my pizza right: Repairing missing is-a relations in ALC ontologies, 2012, Second Joint International Semantic Technology Conference, 17-32.

http://dx.doi.org/10.1007/978-3-642-37996-3_2

Postprint available at: Linköping University Electronic Press

<http://urn.kb.se/resolve?urn=urn:nbn:se:liu:diva-91435>

Get my pizza right: Repairing missing is-a relations in $\mathcal{ALC}$ ontologies

Patrick Lambrix, Zlatan Dragisic, and Valentina Ivanova

Department of Computer and Information Science
and Swedish e-Science Research Centre
Linköping University, 581 83 Linköping, Sweden
`firstname.lastname@liu.se`
<http://www.ida.liu.se/>

Abstract. With the increased use of ontologies in semantically-enabled applications, the issue of debugging defects in ontologies has become increasingly important. These defects can lead to wrong or incomplete results for the applications. Debugging consists of the phases of detection and repairing. In this paper we focus on the repairing phase of a particular kind of defects, i.e. the missing relations in the is-a hierarchy. Previous work has dealt with the case of taxonomies. In this work we extend the scope to deal with $\mathcal{ALC}$ ontologies that can be represented using acyclic terminologies. We present algorithms and discuss a system.

This is a preprint of the paper published by Springer:

Lambrix P, Dragisic Z, Ivanova V, Get my pizza right: Repairing missing is-a relations in $\mathcal{ALC}$ ontologies, Second Joint International Semantic Technology Conference, LNCS 7774, 17-32, Nara, Japan, 2012. *The final publication is available at www.springerlink.com.*
doi: 10.1007/978-3-642-37996-3_2

1 Introduction

Developing ontologies is not an easy task, and often the resulting ontologies are not consistent or complete. Such ontologies, although often useful, also lead to problems when used in semantically-enabled applications. Wrong conclusions may be derived or valid conclusions may be missed. Defects in ontologies can take different forms (e.g. [16]). Syntactic defects are usually easy to find and to resolve. Defects regarding style include such things as unintended redundancy. More interesting and severe defects are the modeling defects which require domain knowledge to detect and resolve, and semantic defects such as unsatisfiable concepts and inconsistent ontologies. Most work up to date has focused on debugging (i.e. detecting and repairing) the semantic defects in an ontology (e.g. [16, 15, 29, 5]). Modeling defects have mainly been discussed in [2, 20, 19] for taxonomies, i.e. from a knowledge representation point of view, a simple kind of

ontologies. The focus has been on defects regarding the is-a structure of the taxonomies. In this paper we tackle the problem of repairing the is-a structure of $\mathcal{ALC}$ ontologies that can be represented using acyclic terminologies.

In addition to its importance for the correct modeling of a domain, the structural information in ontologies is also important in semantically-enabled applications. For instance, the is-a structure is used in ontology-based search and annotation. In ontology-based search, queries are refined and expanded by moving up and down the hierarchy of concepts. Incomplete structure in ontologies influences the quality of the search results. As an example, suppose we want to find articles in the MeSH (Medical Subject Headings [25], controlled vocabulary of the National Library of Medicine, US) Database of PubMed [27] using the term *Scleral Diseases* in MeSH. By default the query will follow the hierarchy of MeSH and include more specific terms for searching, such as *Scleritis*. If the relation between *Scleral Diseases* and *Scleritis* is missing in MeSH, we will miss 738 articles in the search result, which is about 55% of the original result. The structural information is also important information in ontology engineering research. For instance, most current ontology alignment systems use structure-based strategies to find mappings between the terms in different ontologies (e.g. overview in [21]) and the modeling defects in the structure of the ontologies have an important influence on the quality of the ontology alignment results.

Debugging modeling defects in ontologies consists of two phases, detection and repair. There are different ways to detect missing is-a relations. One way is by inspection of the ontologies by domain experts. Another way is to use external knowledge sources. For instance, there is much work on finding relationships between terms in the ontology learning area [3]. In this setting, new ontology elements are derived from text using knowledge acquisition techniques. Regarding the detection of is-a relations, one paradigm is based on linguistics using lexico-syntactic patterns. The pioneering research conducted in this line is in [11], which defines a set of patterns indicating is-a relationships between words in the text. Another paradigm is based on machine learning and statistical methods. Further, guidelines based on logical patterns can be used [5]. When the ontology is part of a network of ontologies connected by mappings between the ontologies, knowledge inherent in the ontology network can be used to detect missing is-a relations using logical derivation [2, 20, 19]. However, although there are many approaches to detect missing is-a relations, these approaches, in general, do not detect *all* missing is-a relations. For instance, although the precision for the linguistics-based approaches is high, their recall is usually very low.

In this paper we assume that the detection phase has been performed. We assume that we have obtained a set of missing is-a relations for a given ontology and focus on the repairing phase. In the case where our set of missing is-a relations contains *all* missing is-a relations, the repairing phase is easy. We just add all missing is-a relations to the ontology and a reasoner can compute all logical consequences. However, when the set of missing is-a relations does not contain all missing is-a relations - and this is the common case - there are different ways to repair the ontology. The easiest way is still to just add the

missing is-a relations to the ontology. For instance, Figure 1 shows a small part of a pizza ontology based on [26], that is relevant for our discussions. Assume that we have detected that $\text{MyPizza} \sqsubset \text{FishyMeatyPizza}$ and $\text{MyFruttiDiMare} \sqsubset \text{NonVegetarianPizza}$ are missing is-a relations. Obviously, adding $\text{MyPizza} \sqsubset \text{FishyMeatyPizza}$ and $\text{MyFruttiDiMare} \sqsubset \text{NonVegetarianPizza}$ to the ontology will repair the missing is-a structure. However, there are other more interesting possibilities. For instance, adding $\text{AnchoviesTopping} \sqsubset \text{FishTopping}$ and $\text{ParmaHamTopping} \sqsubset \text{MeatTopping}$ will also repair the missing is-a structure. Another more informative way¹ to repair the missing is-a structure is to add $\text{AnchoviesTopping} \sqsubset \text{FishTopping}$ and $\text{ParmaHamTopping} \sqsubset \text{HamTopping}$. Essentially, these other possibilities to repair the ontology include missing is-a relations (e.g. $\text{AnchoviesTopping} \sqsubset \text{FishTopping}$) that were not originally detected by the detection algorithm.² We also note that from a logical point of view, adding $\text{AnchoviesTopping} \sqsubset \text{MeatTopping}$ and $\text{ParmaHamTopping} \sqsubset \text{FishTopping}$ also repairs the missing is-a structure. However, from the point of view of the domain, this solution is not correct. Therefore, as it is the case for all approaches for debugging modeling defects, a domain expert needs to validate the logical solutions.

The contributions of this paper are threefold. First, we show that the problem of finding possible ways to repair the missing is-a structure in an ontology in general can be formalized as a generalized version of the TBox abduction problem (Section 3). Second, we propose an algorithm to generate different ways to repair the missing is-a structure in $\mathcal{ALC}$ ontologies that can be represented using acyclic terminologies (Section 4). Third, we discuss a system that allows a domain expert to repair the missing is-a structure in Section 5. We discuss the functionality and user interface of the repairing system and show an example run. Further, we discuss related work in Section 6 and conclude in Section 7. We continue, however, with some necessary preliminaries in Section 2.

2 Preliminaries

In this paper we deal with ontologies represented in the description logic $\mathcal{ALC}$ with acyclic terminologies (e.g. [1]). Concept descriptions are defined using constructors as well as concept and role names. As constructors $\mathcal{ALC}$ allows concept conjunction ($C \sqcap D$), disjunction ($C \sqcup D$), negation ($\neg C$), universal quantification ($\forall r.C$) and existential quantification ($\exists r.C$).³ In this paper we consider ontologies that can be represented by a TBox that is an acyclic terminology. An acyclic terminology is a finite set of concept definitions (i.e. terminological axioms of the form $C \doteq D$ where C is a concept name) that neither con-

¹ This is more informative in the sense that the former is derivable from the latter.

Adding $\text{ParmaHamTopping} \sqsubset \text{HamTopping}$, also allows to derive $\text{ParmaHamTopping} \sqsubset \text{MeatTopping}$ as the ontology already includes $\text{HamTopping} \sqsubset \text{MeatTopping}$.

² Therefore, the approach discussed in this paper can also be seen as a detection method that takes already found missing is-a relations as input.

³ C and D represent concepts, and r represents a role.

Pizza	$\sqsubseteq \top$
PizzaTopping	$\sqsubseteq \top$
hasTopping	$\sqsubseteq \top \times \top$
AnchoviesTopping	$\sqsubseteq \text{PizzaTopping}$
MeatTopping	$\sqsubseteq \text{PizzaTopping}$
HamTopping	$\sqsubseteq \text{MeatTopping}$
ParmaHamTopping	$\sqsubseteq \text{PizzaTopping}$
FishTopping	$\sqsubseteq \text{PizzaTopping} \sqcap \neg \text{MeatTopping}$
TomatoTopping	$\sqsubseteq \text{PizzaTopping} \sqcap \neg \text{MeatTopping} \sqcap \neg \text{FishTopping}$
GarlicTopping	$\sqsubseteq \text{PizzaTopping} \sqcap \neg \text{MeatTopping} \sqcap \neg \text{FishTopping}$
MyPizza	$\doteq \text{Pizza} \sqcap \exists \text{hasTopping}. \text{AnchoviesTopping} \sqcap \exists \text{hasTopping}. \text{ParmaHamTopping}$
FishyMeatyPizza	$\doteq \text{Pizza} \sqcap \exists \text{hasTopping}. \text{FishTopping} \sqcap \exists \text{hasTopping}. \text{MeatTopping}$
MyFruttiDiMare	$\doteq \text{Pizza} \sqcap \exists \text{hasTopping}. \text{AnchoviesTopping}$ $\sqcap \exists \text{hasTopping}. \text{GarlicTopping} \sqcap \exists \text{hasTopping}. \text{TomatoTopping}$ $\sqcap \forall \text{hasTopping}. (\text{AnchoviesTopping} \sqcup \text{GarlicTopping} \sqcup \text{TomatoTopping})$
VegetarianPizza	$\doteq \text{Pizza} \sqcap \neg \exists \text{hasTopping}. \text{FishTopping} \sqcap \neg \exists \text{hasTopping}. \text{MeatTopping}$
NonVegetarianPizza	$\doteq \text{Pizza} \sqcap \neg \text{VegetarianPizza}$

Fig. 1. A pizza ontology.

tains multiple definitions nor cyclic definitions.⁴ An ABox contains assertional knowledge, i.e. statements about the membership of individuals (interpreted as elements in the domain) to concepts $C(i)$ as well as relations between individuals $r(i,j)$.⁵ A knowledge base contains a TBox and an ABox. A model of the TBox/ABox/knowledge base satisfies all axioms of the TBox/ABox/knowledge base. A knowledge base is consistent if it does not contain contradictions.

An important reasoning service is the checking of (un)satisfiability of concepts (a concept is unsatisfiable if it is necessarily interpreted as the empty set in all models of the TBox/ABox/knowledge base, satisfiable otherwise). A TBox is incoherent if it contains an unsatisfiable concept.

Checking satisfiability of concepts in $\mathcal{ALC}$ can be done using a tableau-based algorithm (e.g. [1]). To test whether a concept C is satisfiable such an algorithm starts with an ABox containing the statement $C(x)$ where x is a new individual and it is usually assumed that C is normalized to negation normal form. It then applies consistency-preserving transformation rules to the ABox (Figure 2). The $\sqcap$ -, $\forall$ - and $\exists$ -rules extend the ABox while the $\sqcup$ -rule creates multiple ABoxes. The algorithm continues applying these transformation rules to the ABoxes until no more rules apply. This process is called completion and if one of the final ABoxes does not contain a contradiction (we say that it is open), then satisfiability is proven, otherwise unsatisfiability is proven. One way of implementing this approach is through completion graphs which are directed graphs in which every node represents an ABox. Application of the $\sqcup$ -rule produces new nodes with one statement each, while the other rules add statements to the node on which

⁴ We observe that the TBox in Figure 1 is not an acyclic terminology as there are statements of the form $A \sqsubseteq C$. However, it is possible to create an equivalent TBox that is a acyclic terminology by replacing the statements of the form $A \sqsubseteq C$ with $A \doteq C \sqcap \bar{A}$ where $\bar{A}$ is new atomic concept. See [18].

⁵ i and j represent individuals. In the completion graph in Section 4 statements of the form $C(i)$ are written as $i : C$, and statements of the form $r(i,j)$ are written as irj .

$\sqcap$ -rule:	if the ABox contains $(C_1 \sqcap C_2)(x)$, but it does not contain both $C_1(x)$ and $C_2(x)$, then these are added to the ABox.
$\sqcup$ -rule:	if the ABox contains $(C_1 \sqcup C_2)(x)$, but it contains neither $C_1(x)$ nor $C_2(x)$, then two ABoxes are created representing the two choices of adding $C_1(x)$ or adding $C_2(x)$.
$\forall$ -rule:	if the ABox contains $(\forall r.C)(x)$ and $r(x,y)$, but it does not contain $C(y)$, then this is added to the ABox.
$\exists$ -rule:	if the ABox contains $(\exists r.C)(x)$ but there is no individual z such that $r(x,z)$ and $C(z)$ are in the ABox, then $r(x,y)$ and $C(y)$ with y an individual name not occurring in the ABox, are added.

Fig. 2. Transformation rules (e.g. [1]).

the rule is applied. The ABox for a node contains all the statements of the node as well as the statements of the nodes on the path to the root. Satisfiability is proven if at least one of the ABoxes connected to a leaf node does not contain a contradiction, otherwise unsatisfiability is proven.

In this paper we assume that an ontology O is represented by a knowledge base containing a TBox that is an acyclic terminology and an empty ABox. In this case reasoning can be reduced to reasoning without the TBox by unfolding the definitions. However, for efficiency reasons, instead of running the previously described satisfiability checking algorithm on an unfolded concept description, the unfolding is usually performed on demand within the satisfiability checking algorithm. It has been proven that satisfiability checking w.r.t. acyclic terminologies is PSPACE-complete in $\mathcal{ALC}$ [23].

3 An abduction problem

In our setting, a missing is-a relation in an ontology O represented by a knowledge base KB , is an is-a relation between named concepts that is not derivable from the KB , but that is correct according to the intended domain. We assume that we have a set M of missing is-a relations (but not necessarily all) for O . Then, the is-a structure of O can be repaired by adding is-a relations (or axioms of the form $C \sqsubseteq D$) between named concepts to O such that the missing is-a relations can be derived from the repaired ontology. This repair problem can be formulated as a generalized version of the TBox abduction problem.

Definition 1 *Let KB be a knowledge base in $\mathcal{L}$, and for $1 \leq i \leq m$: C_i, D_i are concepts that are satisfiable w.r.t. KB , such that $KB \cup \{ C_i \sqsubseteq D_i \mid 1 \leq i \leq m \}$ is coherent. A solution to the generalized TBox abduction problem for $(KB, \{(C_i, D_i) \mid 1 \leq i \leq m\})$ is any finite set $S_{GT} = \{G_j \sqsubseteq H_j \mid j \leq n\}$ of TBox assertions in $\mathcal{L}'$ such that $\forall i: KB \cup S_{GT} \models C_i \sqsubseteq D_i$. The set of all such solutions is denoted by $S_{GT}(KB, \{(C_i, D_i) \mid 1 \leq i \leq m\})$.*

In our setting the language $\mathcal{L}$ is $\mathcal{ALC}$ and $\mathcal{L}'$ only allows *named* concepts. We say that any solution in $S_{GT}(KB, \{(C_i, D_i)\}_i)$ is a *repairing action*. A repairing action is thus a set of is-a relations. When $m = 1$, this definition of the generalized TBox abduction problem coincides with the TBox abduction problem as formulated in [8], which therefore deals with repairing one missing is-a relation.

<p>Input: The ontology O represented by knowledge base KB and a set of missing is-a relations M.</p> <p>Output: Set of repairing actions $Rep(M)$.</p> <p>Algorithm</p> <ol style="list-style-type: none"> 1. For every missing is-a relation $A_i \sqsubset B_i$ in M: <ol style="list-style-type: none"> 1.1 G = completion graph after running a tableaux algorithm with $A_i \sqcap \neg B_i$ as input on KB; 1.2 $Leaf-ABoxes$ = get ABoxes of the leaves from the completion graph G; 1.3 For every open ABox $\mathcal{A} \in Leaf-ABoxes$: <ol style="list-style-type: none"> 1.3.1 $R_{\mathcal{A}} = \emptyset$; 1.3.2 For every individual x_j in $\mathcal{A}$: <ol style="list-style-type: none"> 1.3.2.1 $Pos_{x_j} = \{P \mid x_j : P \in \mathcal{A} \wedge P \text{ is a named concept}\}$; 1.3.2.2 $Neg_{x_j} = \{N \mid x_j : \neg N \in \mathcal{A} \wedge N \text{ is a named concept}\}$; 1.3.2.3 $R_{\mathcal{A}} = R_{\mathcal{A}} \cup \{P \sqsubseteq N \mid P \in Pos_{x_j} \wedge N \in Neg_{x_j}\}$; 1.4 $Rep(A_i, B_i) = \emptyset$; 1.5 As long as there are different choices: <ol style="list-style-type: none"> 1.5.1 Create a repairing action ra by choosing one element from each set $R_{\mathcal{A}}$; 1.5.2 $Rep(A_i, B_i) = Rep(A_i, B_i) \cup \{ra\}$; 1.5.3 Remove redundancy in $Rep(A_i, B_i)$; 1.5.4 Remove incoherent solutions from $Rep(A_i, B_i)$; 2. $Rep(M) = \{M\}$; 3. As long as there are different choices: <ol style="list-style-type: none"> 3.1 Create a repairing action rp by choosing one element from each $Rep(A_i, B_i)$ and taking the union of these elements; 3.2 $Rep(M) = Rep(M) \cup \{rp\}$; 4. Remove redundancy in $Rep(M)$; 5. Remove incoherent solutions from $Rep(M)$;
--

Fig. 3. Basic algorithm for generating repairing actions using completion graph.

Further, we have that $S_{GT}(KB, \{(C_i, D_i)\}_i) = \cap_i S_{GT}(KB, \{(C_i, D_i)\})$.⁶ This shows that solving a generalized TBox abduction problem can be done by solving m TBox abduction problems and then taking the intersection of the solutions. In practice, however, this leads to a number of difficulties. First, it would mean that a domain expert will need to choose between large sets of repairing actions for all the missing is-a relations at once, and this may be a very hard task. Further, due to the size of the solution space, even generating all solutions for one TBox abduction problem is, in general, infeasible. Also, many of the solutions will not be interesting for a domain expert (e.g. [20]). For instance, there may be solutions containing is-a relations that do not contribute to the actual repairing. Some solutions may introduce unintended equivalence relations. A common way to limit the number of possible solutions is to introduce constraints, e.g. minimality.

4 Algorithm for generating repairing actions

Basic algorithm The basic algorithm for generating repairing actions for a set of given missing is-a relations for an ontology is shown in Figure 3. In this

⁶ A solution for all missing is-a relations is also a solution for each missing is-a relation and therefore, in the intersection of the solutions concerning one missing is-a relation at the time. Further, a solution in the intersection of the sets of solutions for each of the missing is-a relations, allows, when added to the knowledge base, to derive all missing is-a relations, and is therefore, a solution for the generalized TBox abduction problem as well.

first study we assume that the existing structure in the ontology is correct. Also, as stated in the definition in Section 3, adding the missing is-a relations to the ontology does not lead to incoherence.

In step 1 a set of repairing actions is generated for each missing is-a relation (thereby solving a TBox abduction problem for each missing is-a relation). In step 1.1 we run the satisfiability checking algorithm with unfolding on demand as described in Section 2, on KB with input $A_i \sqcap \neg B_i$, and we collect the ABoxes of the leaves in step 1.2. As $A_i \sqsubset B_i$ is a missing is-a relation, it cannot be derived from KB and thus the completion graph will have open leaf ABoxes. We then generate different ways to close these ABoxes in step 1.3. For each individual x in an open leaf ABox we collect the concepts in the statements of the form $x:A$ in Pos_x and the concepts in the statements of the form $x:\neg B$ in Neg_x where A and B are named concepts. The ABox can be closed if $A \sqsubset B$ is added to the ontology for any $A \in Pos_x$ and $B \in Neg_x$. Indeed, with this extra information $x:A$ could be unfolded and $x:B$ would be added to the ABox, and this gives a contradiction with $x:\neg B$ which was already in the ABox. A repairing action for the missing is-a relation is then a set of $A \sqsubset B$ that closes each open leaf ABox. In step 1.5 we generate such sets by selecting one such axiom per open leaf ABox and remove redundancy based on the sub-set relation. If a repairing action is a super-set of another repairing action, it is removed. Further, we remove solutions that introduce incoherence.

In step 2 the repairing actions set for all missing is-a relations is initialized with the set of missing is-a relations. Therefore, there will always be at least one repairing action. In step 3 additional repairing actions are generated by combining repairing actions for the individual missing is-a relations. As repairing actions are sets, there are no duplicates in a repairing action. In step 4 we remove redundancy based on the sub-set relation. In step 5 we remove solutions that introduce incoherence. We note that there always will be at least one solution that does not introduce incoherence (i.e. M or a sub-set of M).

A repairing action is a set of statements of the form $A \sqsubset B$ with A and B named concepts. We note that for acyclic terminologies, after adding such a statement, the resulting TBox is not an acyclic terminology anymore. If this is needed, then instead of adding $A \sqsubset B$ the following should be done (see [18]). If there is no definition for A yet in the TBox, then add $A \doteq B \sqcap \bar{A}$ with $\bar{A}$ a new atomic concept. If there is already a definition for A in the TBox, say $A \doteq C$ then change this definition to $A \doteq B \sqcap C$.

Our proposed algorithm is sound and it generates solutions to a TBox abduction problem that are minimal in the sense that repairing actions only contain necessary information for repairing the missing is-a relations. Further, we check the generated solutions for the introduction of incoherence.⁷

As an example, consider the acyclic terminology equivalent to the ontology in Figure 1 and $M = \{\text{MyPizza} \sqsubset \text{FishyMeatyPizza}, \text{MyFruttiDiMare} \sqsubset \text{Non-VegetarianPizza}\}$. For the missing is-a relation $\text{MyPizza} \sqsubset \text{FishyMeatyPizza}$ the completion graph obtained after running the satisfiability check on $\text{MyPizza} \sqcap$

⁷ For the proof of the properties of the algorithm we refer to [18].

$\neg \text{FishyMeatyPizza}$ is shown in Figure 4. The completion graph contains 17 nodes of which 11 are leaf nodes. Of these leaf nodes 6 are closed and the repairing actions will be based on the 5 open leaf nodes. Closing all open leaf ABoxes will lead to 11 non-redundant repairing actions of which 8 lead to incoherence. The remaining repairing actions are $\{\text{MyPizza} \sqsubseteq \text{FishyMeatyPizza}\}$, $\{\text{AnchoviesTopping} \sqsubseteq \text{FishTopping}, \text{ParmaHamTopping} \sqsubseteq \text{MeatTopping}\}$ and $\{\text{ParmaHamTopping} \sqsubseteq \text{FishTopping}, \text{AnchoviesTopping} \sqsubseteq \text{MeatTopping}\}$. In a similar way, for missing is-a relation $\text{MyFruttiDiMare} \sqsubseteq \text{NonVegetarianPizza}$, we find the following non-redundant coherence-preserving repairing actions: $\{\text{MyFruttiDiMare} \sqsubseteq \text{NonVegetarianPizza}\}$, $\{\text{AnchoviesTopping} \sqsubseteq \text{FishTopping}\}$ and $\{\text{AnchoviesTopping} \sqsubseteq \text{MeatTopping}\}$. After combining the repairing actions for the individual missing is-a relations, and removing redundancy and incoherence-introducing repairing actions, we obtain the following 5 solutions for the repairing of the missing is-a structure: $\{\text{MyPizza} \sqsubseteq \text{FishyMeatyPizza}, \text{MyFruttiDiMare} \sqsubseteq \text{NonVegetarianPizza}\}$, $\{\text{AnchoviesTopping} \sqsubseteq \text{FishTopping}, \text{ParmaHamTopping} \sqsubseteq \text{MeatTopping}\}$, $\{\text{ParmaHamTopping} \sqsubseteq \text{FishTopping}, \text{AnchoviesTopping} \sqsubseteq \text{MeatTopping}\}$, $\{\text{MyPizza} \sqsubseteq \text{FishyMeatyPizza}, \text{AnchoviesTopping} \sqsubseteq \text{FishTopping}\}$, and $\{\text{MyPizza} \sqsubseteq \text{FishyMeatyPizza}, \text{AnchoviesTopping} \sqsubseteq \text{MeatTopping}\}$.

Optimization The basic algorithm may produce many redundant solutions already in step 1. For instance, if the root node in the completion graph contains $x:A$ and $x:\neg B$ with A and B named concepts, then $A \sqsubseteq B$ appears in every open leaf ABox as a possible way of closing that ABox. To deal with this issue we modify the algorithm such that at each node with an open ABox it generates Pos_{x_j} and Neg_{x_j} . However, Pos_{x_j} and Neg_{x_j} only contain the concepts that are not already in a Pos_{x_j} and Neg_{x_j} related to an ancestor node. For instance, if $A \in Pos_x$ related to the root node, then A will not appear in any Pos_x related to another node. For each ABox we then generate a set $R_A = \bigcup_x \{P \sqsubseteq N\}$ where $P \in Pos_x$ for the current node or any ancestor node, $N \in Neg_x$ for the current node or any ancestor node, and at least one of P and N occurs in the current node. This allows us to reduce the redundancy in step 1. An open leaf ABox can now be closed by using an element from R_A from the leaf node or from any ancestor node. When generating repairing actions in step 1.5 we then make sure that when an element related to a non-leaf node is chosen, that no additional element from any descendant of that node is selected to close any leaf ABoxes that are descendants of that non-leaf node. For instance, if any element from the root's R_A is chosen, then no other elements should be chosen. For an example see [18].

Extension The algorithm can be extended to generate additional repairing actions for every individual missing is-a relation. In step 1.5 if $A \sqsubseteq B$ is used as one of the is-a relations in a repairing action then also $S \sqsubseteq T$ where S is a super-concept of A and T is a sub-concept of B could be used. Therefore, the extended algorithm generates two sets of concepts for every is-a relation $A \sqsubseteq$

Fig. 5. Screenshot - Validating is-a relations in a repairing action.

B in a repairing action, Source set containing named super-concepts of A and Target set containing named sub-concepts of B . Further, to not introduce non-validated equivalence relations where in the original ontology there are only is-a relations, we remove the super-concepts of B from Source, and the sub-concepts of A from Target. Alternative repairing actions for a repairing action $\{A_1 \sqsubseteq B_1, \dots, A_n \sqsubseteq B_n\}$ are then repairing actions $\{S_1 \sqsubseteq T_1, \dots, S_n \sqsubseteq T_n\}$ such that $(S_i, T_i) \in Source(A_i, B_i) \times Target(A_i, B_i)$.

5 Implementation

We have implemented a system that supports the user to repair missing is-a relations. In our system the user loads the ontology and the missing is-a relations from the 'File' menu. The missing is-a relations are then shown in a drop-down list (e.g. $MyPizza \sqsubseteq FishyMeatyPizza$ ⁸ in the second drop-down list in Figure 5). The user then chooses a missing is-a relation to repair.⁹ At any time the user can switch between different missing is-a relations. Once a missing is-a relation is chosen for repairing the user generates repairing actions for it by clicking the

⁸ In the system $C \sqsubseteq D$ is shown as $C \rightarrow D$.

⁹ As we repair one is-a relation at a time, there may be some redundancy in the solutions.

Fig. 6. Screenshot - Repairing using Source and Target sets.

'Generate Repairing Actions' button. This covers step 1 in Figure 3 and was implemented in Java using Pellet (version 2.3.0) [31]. The satisfiability checker in Pellet was modified in order to extract full completion graphs. Furthermore, to increase performance and account for higher level of non-determinism, ontologies are first passed through Pellint [22] before running the algorithm. The computed repairing actions are shown in the drop-down list under the button. Each repairing action consists of one or more is-a relations. In Figure 5 the user has chosen to repair $\text{MyPizza} \sqsubseteq \text{FishyMeatyPizza}$ and the system has generated three repairing actions that do not introduce incoherence ($\{\text{MyPizza} \sqsubseteq \text{FishyMeatyPizza}\}$, $\{\text{AnchoviesTopping} \sqsubseteq \text{FishTopping}, \text{ParmaHamTopping} \sqsubseteq \text{MeatTopping}\}$, and $\{\text{AnchoviesTopping} \sqsubseteq \text{MeatTopping}, \text{ParmaHamTopping} \sqsubseteq \text{FishTopping}\}$). To repair this missing is-a relation the user has to successfully deal with at least one of the repairing actions, i.e. add all is-a relations (or more informative is-a relations from Source and Target) in at least one of the repairing actions to the ontology. In Figure 5 the user has chosen the second repairing action. When repairing actions are added to the ontology, they will make the missing is-a relation derivable. However, it is not guaranteed that all the is-a relations in the repairing actions are also valid with respect to the domain. Therefore, a domain expert needs to validate the is-a relations in the repairing actions. When the user clicks the 'Validate is-a Relations in Repairing Action' button, a pop-up window (Figure 5) appears where the user can mark all the

is-a relations that are correct with respect to the domain model. The repairing actions for all missing is-a relations and the ontology are updated according to the results of the validation. If an is-a relation is validated as incorrect according to the domain, all repairing actions that contain this incorrect is-a relation, for this and for all other missing is-a relations, are removed from the lists of the repairing actions. When an is-a relation is validated as correct it is added to the ontology and it is marked as correct in all repairing actions for all missing is-a relations. When all is-a relations in the current repairing action are validated as correct, they are shown in the last drop-down list (Figure 6). Now the user can repair them one by one.

For each is-a relation within the repairing action the Source and Target sets are generated and displayed on the left and the right hand sides, respectively, within the panel under the drop-down lists (Figure 6). Both panels have zoom control and can be opened in a separate window. The concepts in the is-a relation under consideration are highlighted in red, existing asserted and inferred is-a relations are shown in grey, not yet repaired missing is-a relations in blue and is-a relations that were previously added for repairing missing is-a relations in black. In order to repair the is-a relation the user has to choose one concept from each of the sets and click the 'Repair' button. In Figure 6 the user has chosen to repair $\text{ParmaHamTopping} \sqsubseteq \text{MeatTopping}$ with $\text{ParmaHamTopping} \sqsubseteq \text{HamTopping}$. Upon repair the ontology is updated, i.e. the chosen is-a relation ($\text{ParmaHamTopping} \sqsubseteq \text{HamTopping}$) is added to the ontology. A red label next to the drop-down list shows the status (Repaired or Not Repaired) of the selected is-a relation. When all is-a relations within a repairing action are repaired the missing is-a relation is marked as repaired ('This relation is already repaired' label in Figure 6). The other repairing actions are still available for review by the user. These may give information about other possible missing is-a relations. The user can also revoke repairing actions (through the 'Revoke Repairing Actions' button). If the user revokes the repairing action, the missing is-a relation may become not repaired again and the is-a relations within the repairing action are marked as not repaired. All changes in the ontology are revoked and the user can start repairing this missing is-a relation again in the way just described.

Example run As an example run, consider the ontology in Figure 1 and missing is-a relations $\text{MyPizza} \sqsubseteq \text{FishyMeatyPizza}$ and $\text{MyFruttiDiMare} \sqsubseteq \text{NonVegetarianPizza}$. After loading the ontology and the missing is-a relations, we can choose a missing is-a relation to repair. Assume we choose $\text{MyFruttiDiMare} \sqsubseteq \text{NonVegetarianPizza}$ and click the 'Generate Repairing Actions' button. The system will show three repairing actions in the drop-down list: $\{\text{MyFruttiDiMare} \sqsubseteq \text{NonVegetarianPizza}\}$, $\{\text{AnchoviesTopping} \sqsubseteq \text{FishTopping}\}$, and $\{\text{AnchoviesTopping} \sqsubseteq \text{MeatTopping}\}$. We can choose to deal with $\text{AnchoviesTopping} \sqsubseteq \text{MeatTopping}$ and validate this to be incorrect with respect to the domain. In this case all repairing actions containing this is-a relation will be removed. We could then choose $\text{AnchoviesTopping} \sqsubseteq \text{FishTopping}$ and validate it to be correct. All is-a relations in this repairing action (i.e. $\text{AnchoviesTopping} \sqsubseteq \text{FishTopping}$) are validated to be correct and thus we can continue with the repair of AnchoviesTop-

ping $\sqsubseteq$ FishTopping. In this small example the Source set only contains AnchoviesTopping and the Target set only contains FishTopping. Therefore, we click on these concepts and the 'Repair' button. AnchoviesTopping $\sqsubseteq$ FishTopping will be marked as repaired and also the missing is-a relation MyFruttiDiMare $\sqsubseteq$ NonVegetarianPizza will be marked as repaired.

We can then start repairing MyPizza $\sqsubseteq$ FishyMeatyPizza. The system would have generated as repairing actions that do not introduce incoherence $\{\text{MyPizza} \sqsubseteq \text{FishyMeatyPizza}\}$, $\{\text{AnchoviesTopping} \sqsubseteq \text{FishTopping}, \text{ParmaHamTopping} \sqsubseteq \text{MeatTopping}\}$ and $\{\text{AnchoviesTopping} \sqsubseteq \text{MeatTopping}, \text{ParmaHamTopping} \sqsubseteq \text{FishTopping}\}$. However, as we earlier already validated AnchoviesTopping $\sqsubseteq$ MeatTopping to be incorrect with respect to the domain, the third repairing action has already been removed. When we choose the second repairing action, AnchoviesTopping $\sqsubseteq$ FishTopping is already marked as correct (because of earlier validation) and only ParmaHamTopping $\sqsubseteq$ MeatTopping needs to be validated. We validate this as correct and then choose to repair it. The Source set in this small example contains ParmaHamTopping and the Target set contains HamTopping and MeatTopping. Although we can add ParmaHamTopping $\sqsubseteq$ MeatTopping, it is more informative (and correct with respect to the domain) to add ParmaHamTopping $\sqsubseteq$ HamTopping. We therefore choose the latter. All is-a relations in this repairing action are then repaired and thus also MyPizza $\sqsubseteq$ FishyMeatyPizza.

6 Related work

Debugging ontologies *Detecting* missing is-a relations can be done in a number of ways (see Section 1). There is, however, not much work on the *repairing* of missing is-a structure. In [20] we addressed this in the setting of taxonomies.

There is more work on the debugging of semantic defects. Most of it aims at identifying and removing logical contradictions from an ontology. Standard reasoners are used to identify the existence of a contradiction, and provide support for resolving and eliminating it [9]. In [29] minimal sets of axioms are identified which need to be removed to render an ontology coherent. In [16, 15] strategies are described for repairing unsatisfiable concepts detected by reasoners, explanation of errors, ranking erroneous axioms, and generating repair plans. In [10] the focus is on maintaining the consistency as the ontology evolves through a formalization of the semantics of change for ontologies. In [24] and [13] the setting is extended to repairing ontologies connected by mappings. In this case, semantic defects may be introduced by integrating ontologies. Both works assume that ontologies are more reliable than the mappings and try to remove some of the mappings to restore consistency. The solutions are often based on the computation of minimal unsatisfiability-preserving sets or minimal conflict sets. The work in [28] further characterizes the problem as mapping revision. Using belief revision theory, the authors give an analysis for the logical properties of the revision algorithms. Another approach for debugging mappings is proposed in [32] where the authors focus on the detection of certain kinds of defects and

redundancy. The approach in [14] deals with the inconsistencies introduced by the integration of ontologies, and unintended entailments validated by the user.

Work that deals with both modeling and semantic defects includes [5] where the authors propose an approach for detecting modeling and semantic defects within an ontology based on patterns and antipatterns. Some suggestions for repairing are also given. In [19] we provided a method to detect and repair wrong and missing is-a structure in taxonomies connected by mappings.

A different setting is the area of modular ontologies where the ontologies are connected by directional mappings and where knowledge propagation only occurs in one direction. Regarding the detection of semantic defects, within a framework based on distributed description logics, it is possible to restrict the propagation of local inconsistency to the whole set of ontologies (e.g. [30]).

Abductive reasoning in description logics In [8] four different abductive reasoning tasks are defined - (conditionalized) concept abduction, ABox abduction, TBox abduction and knowledge base abduction. Concept abduction deals with finding sub-concepts. Abox abduction deals with retrieving abductively instances of concepts or roles that, when added to the knowledge base, allow the entailment of a desired ABox assertion. Knowledge base abduction includes both ABox and TBox abduction.

Most existing approaches for DL abduction focus on ABox and concept abduction and are mostly based on existing proof techniques such as semantic tableaux and resolution. Since the number of possible solutions is very large, the approaches introduce constraints. The work in [17] proposes a goal-oriented approach where only actions which contribute to the solution are chosen in the proof procedures. The method is both complete and sound for consistent and semantically minimal solutions. Since the set of solutions can contain some inconsistent and non-minimal solutions additional checks are required. A practical approach for ABox abduction, based on abductive logic programming, was proposed in [7]. In order to use existing abductive logic programming systems it is necessary to do a transformation to a plain Datalog program. The solutions are consistent and minimal given a set of abducibles. However, the approach does not guarantee completeness since the translation to a Datalog program is approximate and in some cases a solution would not be found. An approach for conditionalised concept abduction that uses a variation of the semantic tableaux and two labeling functions was proposed in [4]. The two labeling functions $T()$ and $F()$ represent true and false formulas in a tableaux where the solutions are formed from concepts which have at least one constraint in $F()$ of every open branch. This choice is non-deterministic and can be used to select solutions based on some criteria. The algorithm also contains a consistency check which implies that produced solutions are always consistent.

There has not been much work related to TBox abduction, which is the most relevant abduction problem for this paper. The work in [12] proposes an automata-based approach to TBox abduction using abducibles representing axioms that can appear in solutions. It is based on a reduction to the axiom pin-

pointing problem which is then solved with automata-based methods. A PTIME algorithm is proposed for the language $\mathcal{EL}$.

All of the presented approaches to description logic abduction work with relatively inexpressive ontologies, such as $\mathcal{EL}$ and $\mathcal{ALC}$. However, some recent work [6] describes a type of conditionalised concept abduction called structural abduction which is applicable to $\mathcal{SH}$.

7 Conclusion

This paper formalized repairing the is-a structure in $\mathcal{ALC}$ acyclic terminologies as a generalized TBox abduction problem, provided a solution based on semantic tableaux, and discussed a system.

There are a number of interesting aspects for future work. First, we intend to extend the algorithm to deal with more expressive ontologies. Further, it may be useful to consider also solutions introducing incoherence as they may lead to the detection of other kinds of modeling defects such as wrong is-a relations. In this case we do not assume anymore that the existing structure is correct. As a domain expert may need to deal with many possible solutions, other useful extensions are mechanisms for ranking the generated repairing actions and among those recommending repairing actions, e.g. based on domain knowledge.

Acknowledgements. We acknowledge the financial support of the Swedish Research Council (Vetenskapsrådet), the Swedish National Graduate School in Computer Science (CUGS) and the Swedish e-Science Research Centre (SeRC).

References

1. F Baader and U Sattler. An overview of tableau algorithms for description logics. *Studia Logica*, 69:5–40, 2001.
2. M Bada and L Hunter. Identification of OBO nonalignments and its implication for OBO enrichment. *Bioinformatics*, 24(12):1448–1455, 2008.
3. Ph Cimiano, P Buitelaar, and B Magnini. *Ontology Learning from Text: Methods, Evaluation and Applications*. IOS Press, 2005.
4. S Colucci, T Di Noia, E Di Sciascio, F Donini, and M Mongiello. A uniform tableaux-based approach to concept abduction and contraction in ALN. In *International Workshop on Description Logics*, pages 158–167, 2004.
5. O Corcho, C Roussey, L M Vilches, and I Pérez. Pattern-based owl ontology debugging guidelines. In *Workshop on Ontology Patterns*, pages 68–82, 2009.
6. F Donini, S Colucci, T Di Noia, and E Di Sciascio. A tableaux-based method for computing least common subsumers for expressive description logics. In *21st International Joint Conference on Artificial Intelligence*, pages 739–745, 2009.
7. J Du, G Qian, Y-D Shen, and J Pan. Towards practical abox abduction in large owl dl ontologies. In *25th AAAI Conference on Artificial Intelligence*, pages 1160–1165, 2011.
8. C Elsenbroich, O Kutz, and U Sattler. A case for abductive reasoning over ontologies. In *OWL: Experiences and Directions*, 2006.

9. G Flouris, D Manakanatas, H Kondylakis, D Plexousakis, and G Antoniou. Ontology Change: Classification and Survey. *Knowledge Engineering Review*, 23(2):117–152, 2008.
10. P Haase and L Stojanovic. Consistent Evolution of OWL Ontologies. In *2nd European Semantic Web Conference*, volume 3532 of *LNCS*, pages 182–197, 2005.
11. M Hearst. Automatic acquisition of hyponyms from large text corpora. In *14th International Conference on Computational Linguistics*, pages 539–545, 1992.
12. T Hubauer, S Lamparter, and M Pirker. Automata-based abduction for tractable diagnosis. In *International Workshop on Description Logics*, pages 360–371, 2010.
13. Q Ji, P Haase, G Qi, P Hitzler, and S Stadtmüller. RaDON - repair and diagnosis in ontology networks. In *6th European Semantic Web Conference*, volume 5554 of *LNCS*, pages 863–867, 2009.
14. E Jimenez-Ruiz, B Cuenca Grau, I Horrocks, and R Berlanga. Ontology Integration Using Mappings: Towards Getting the Right Logical Consequences. In *6th European Semantic Web Conference*, volume 5554 of *LNCS*, pages 173–187, 2009.
15. A Kalyanpur, B Parsia, E Sirin, and B Cuenca-Gray. Repairing Unsatisfiable Concepts in OWL Ontologies. In *3rd European Semantic Web Conference*, volume 4011 of *LNCS*, pages 170–184, 2006.
16. A Kalyanpur, B Parsia, E Sirin, and J Hendler. Debugging Unsatisfiable Classes in OWL Ontologies. *Journal of Web Semantics*, 3(4):268–293, 2006.
17. S Klarman, U Endriss, and S Schlobach. Abox abduction in the description logic ALC. *Journal of Automated Reasoning*, 46:43–80, 2011.
18. P Lambrix, Z Dragisic, and V Ivanova. Get my pizza right: Repairing missing is-a relations in ALC ontologies (extended version). 2012. Available at <http://arxiv.org/abs/1210.7154>.
19. P Lambrix and Q Liu. Debugging is-a structure in networked taxonomies. In *4th International Workshop on Semantic Web Applications and Tools for Life Sciences*, pages 58–65, 2011.
20. P Lambrix, Q Liu, and H Tan. Repairing the Missing is-a Structure of Ontologies. In *4th Asian Semantic Web Conference*, volume 5926 of *LNCS*, pages 76–90, 2009.
21. P Lambrix, L Strömbäck, and H Tan. Information Integration in Bioinformatics with Ontologies and Standards. In Bry and Maluszynski, editors, *Semantic Techniques for the Web: The REWERSE perspective*, chapter 8, pages 343–376. Springer, 2009.
22. H Lin and E Sirin. Pellint - a performance lint tool for pellet. In *OWL: Experiences and Directions*, 2008.
23. C Lutz. Complexity of terminological reasoning revisited. In *16th International Conference on Logic for Programming and Automated Reasoning*, pages 181–200, 1999.
24. C Meilicke, H Stuckenschmidt, and A Tamin. Repairing Ontology Mappings. In *22th Conference on Artificial Intelligence*, pages 1408–1413, 2007.
25. MeSH. Medical subject headings. <http://www.nlm.nih.gov/mesh/>.
26. Pizza Ontology v1.5. <http://www.co-ode.org/ontologies/pizza/2007/02/12/>.
27. PubMed. <http://www.ncbi.nlm.nih.gov/pubmed/>.
28. G Qi, Q Ji, and P Haase. A Conflict-Based Operator for Mapping Revision. In *8th International Semantic Web Conference*, volume 5823 of *LNCS*, pages 521–536, 2009.
29. S Schlobach. Debugging and Semantic Clarification by Pinpointing. In *2nd European Semantic Web Conference*, volume 3532 of *LNCS*, pages 226–240, 2005.

30. L Serafini, A Borgida, and A Tamin. Aspects of distributed and modular ontology reasoning. In *19th International Joint Conference on Artificial Intelligence*, pages 570–575, 2005.
31. E Sirin, B Parsia, B Cuenca Grau, A Kalyanpur, and Y Katz. Pellet: A practical owl-dl reasoner. *Journal of Web Semantics*, 5(2):151–53, 2007.
32. P Wang and B Xu. Debugging ontology mappings: a static approach. *Computing and Informatics*, 27:21–36, 2008.