Ekaterina Ovchinnikova, Kai-Uwe Kühnberger

Automatic Ontology Extension: Resolving Inconsistencies

Ontologies are widely used in text technology and artificial intelligence. The need to develop large ontologies for real-life applications provokes researchers to automate ontology extension procedures. Automatic updates without the control of a human expert can generate potential conflicts between original and new knowledge. As a consequence the resulting ontology can yield inconsistencies. We propose a procedure that models the process of adapting an ontology to new information by repairing several important types of inconsistencies.

1 Introduction

There is an increasing interest in augmenting text technological and artificial intelligence applications with ontological knowledge. Since the manual development of large ontologies has been proven to be time-consuming, many current investigations are devoted to automatic ontology learning methods (Perez and Mancho, 2003).

The most important existing markup language for ontology design is the Web Ontology Language[1] (OWL), with its popular versions (OWL Lite and OWL DL) based on the logical formalism called Description Logic (DL). DL was designed for the representation of terminological knowledge and reasoning devices (Baader et al., 2003). Although most of the tools extracting or extending ontologies automatically output the knowledge in the OWL-format, they usually use only a small subset of the corresponding DL representation. The core ontologies generated in most practical applications contain the subsumption relation defined on concepts (taxonomy) and a few general relations (such as part-of and other). At present complex ontologies making use of the full expressive power and advances of the various versions of DL can be achieved only manually or semi-automatically. However, several recent approaches not only attempt to learn taxonomic and general relations, but also state which concepts in the knowledge base are equivalent or disjoint (Haase and Stojanovic, 2005).

The storage of ontological information within a logical framework entails inconsistency problems, because pieces of information can contradict each other, making the given ontology unsatisfiable and therefore useless for reasoning purposes. The problem of inconsistency becomes even more important with regard to large-scale ontologies: resolving inconsistencies in large ontologies by hand is time-consuming and tedious, therefore automatic procedures to debug ontologies are required.

The approach presented in this paper focuses on logical inconsistencies in terminological knowledge base: after a rough sketch of DLs (Section 2), we discuss informally

[1] A documentation can be found at `http://www.w3.org/TR/owl-features/`.

inconsistencies in ontologies (Section 3) and related work (Section 4). In addition to extending some existing ontology debugging methods, we provide formal criteria to distinguish different types of logical inconsistencies (overgeneralization and polysemy) in Section 5 and present an adaptation algorithm resolving logical inconsistencies that may appear in ontology extensions (Section 6). Section 7 adds some remarks concerning the order of the update and Section 8 concludes the paper.

2 Description Logic

In this section, we define description logics (DL) underlying the ontology representation considered in this paper[2] (cf. Baader et al., 2003, for an overview). A DL ontology contains a set of terminological axioms (called *TBox*), a set of instantiated concepts (called *Assertion* or *ABox*), and a set of role axioms (called *RBox*). In the present paper, we focus on the *TBox*, leaving the *ABox* and the *RBox* aside for further investigation.

A TBox is a finite set of axioms of the form $A_1 \equiv A_2$ (equalities) or $A \sqsubseteq C$ (inclusions), where A stands for a concept name and C (called *concept description*[3]) is defined as follows (R denotes a role name, A denotes an atomic concept): $C \rightarrow A \mid \neg A \mid \forall R.A$.

The semantics of concepts and axioms is defined in the usual way in terms of a model theoretic *interpretation function* $\mathcal{I} = (\Delta^{\mathcal{I}}, \cdot^{\mathcal{I}})$, where $\Delta^{\mathcal{I}}$ is a non-empty set of individuals and the function $\cdot^{\mathcal{I}}$ maps every concept name A to $A^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}}$ and every role name R to $R^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times \Delta^{\mathcal{I}}$. Negation and universal restriction is defined as usual: $(\neg A)^{\mathcal{I}} = \Delta^{\mathcal{I}} \backslash A^{\mathcal{I}}$ and $(\forall R.A)^{\mathcal{I}} = \{x \mid \forall y.\langle x, y\rangle \in R^{\mathcal{I}} \rightarrow y \in A^{\mathcal{I}}\}$. An interpretation $\mathcal{I}$ is a *model* of a TBox $\mathcal{T}$, if for every inclusion $A \sqsubseteq C$ in the TBox, $A^{\mathcal{I}} \subseteq C^{\mathcal{I}}$ and for every equality $A_1 \equiv A_2$ in the TBox, $A_1^{\mathcal{I}} = A_2^{\mathcal{I}}$ holds. A concept description D *subsumes* C in $\mathcal{T}$ ($\mathcal{T} \models C \sqsubseteq D$), if for every model $\mathcal{I}$ of $\mathcal{T}$: $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$. A concept C is called *satisfiable towards* $\mathcal{T}$, if there is a model $\mathcal{I}$ of $\mathcal{T}$ such that $C^{\mathcal{I}}$ is nonempty. Otherwise C is *unsatisfiable towards* $\mathcal{T}$. The algorithms for checking satisfiability of concept descriptions are described in Baader et al. (2003) and implemented in several reasoners[4]. A TBox $\mathcal{T}$ is called *unsatisfiable* iff there is an atomic concept A defined in $\mathcal{T}$ that is unsatisfiable.

An important DL concept for this paper is the *least common subsumer (lcs)* (cf. Baader and Küsters, 2006) for the definition and algorithms for computing lcs). Intuitively, the lcs for concept descriptions C_1 and C_2 is a concept description that collects all common features of C_1 and C_2 and is most specific towards subsumption.

Definition 1 *A concept description L is a least common subsumer (lcs) of concept descriptions $C_1, ..., C_n$ towards a TBox $\mathcal{T}$ iff it satisfies the following two conditions:*

1. $\forall i \in \{1, ..., n\} : \mathcal{T} \models C_i \sqsubseteq L$ *and*
2. $\forall L'$*: if* $\forall i \in \{1, ..., n\} : \mathcal{T} \models C_i \sqsubseteq L'$ *and* $L' \neq L$ *then* $\mathcal{T} \not\models L' \sqsubseteq L$.

[2] In the following definitions, we closely follow Haase and Stojanovic (2005) who present an approach using one of the most powerful DL-versions for ontology learning.

[3] Hereinafter concept descriptions are referred to as concepts.

[4] Some of the DL reasoners are listed at `http://www.cs.man.ac.uk/~sattler/reasoners.html`.

3 Inconsistent Ontologies

The notion of an *inconsistent ontology* has several meanings. For example, three types of inconsistencies are distinguished in Haase and Stojanovic (2005):

- *Structural inconsistency* is defined with respect to the underlying ontology language. An ontology is structurally inconsistent, if it is syntactically inconsistent, i.e. if it contains axioms violating syntactical rules of the representation language (e.g. OWL *DL*).
- *Logical inconsistency* of an ontology is defined on the basis of formal semantics: An ontology is logically inconsistent, if it has no model.
- *User-defined inconsistency* is related to application context constraints defined by the user.

In this paper, we consider logical inconsistencies only. In particular, we focus on unsatisfiable terminologies. Notice that an ontology can become logically inconsistent only if its underlying logic allows negation. Ontologies share this property with every logical system. For the approaches concerned with core ontologies (lacking negation) contradictions in the ontological knowledge base cannot arise. But for approaches using more powerful logics, the problem of inconsistency becomes important (Haase and Stojanovic, 2005).

Terminological unsatisfiability can have several reasons: first, errors in the automatic ontology learning procedure or mistakes of the ontology engineer, second, polysemy of concept names, and third, generalization mistakes. The polysemy problem is particularly relevant for automatic ontology learning. If an ontology is learned automatically, then it is hardly possible to distinguish between word senses: If, for example, a concept *Tree* is declared to be a subconcept of both, *Plant* and *Data structure* (where *Plant* and *Data structure* are subconcepts of disjoint concepts, e.g. *Object* and *Abstraction*), then *Tree* will be unsatisfiable.

Generalization mistakes causing unsatisfiability are connected with definitions of some concepts that are too specific: such definitions contradict with their subconcepts, representing exceptions to these definitions. Here is a classical example:

Example 1

TBox: $\{1.\ Bird \sqsubseteq CanFly,\quad 2.\ CanFly \sqsubseteq CanMove,$
$3.\ Canary \sqsubseteq Bird,\quad 4.\ Penguin \sqsubseteq Bird\}$

New axiom: $\{5.\ Penguin \sqsubseteq \neg CanFly\}$

The statement *all birds can fly* in Example 1 is too specific. If an exception *penguin*, that cannot fly, is added, the terminology becomes unsatisfiable.

Example 2 below demonstrates a case where two overgeneralized definitions of the same concept conflict with each other:

Example 2
TBox: $\{1.\ Child \sqsubseteq \forall likes.Icecream, 2.\ Icecream \sqsubseteq Sweetie, 3.\ Chocolate \sqsubseteq Sweetie,$
$4.\ Icecream \sqsubseteq \neg Chocolate, 5.\ Chocolate \sqsubseteq \neg Icecream\}$
New axiom: $6.\ Child \sqsubseteq \forall likes.Chocolate$

In this example, both definitions of $Child$ (i.e. $\forall likes.Icecream$ and $\forall likes.Chocolate$) are too specific. $Icecream$ and $Chocolate$ being disjoint concepts produce a conflict, if a modeled child likes at least one of their instances. Strictly speaking, the TBox in Example 2 is satisfiable. But we consider contradictions in scopes of universal quantification also as problematic, since such definitions are unusable in practice.

4 Related Work

A technique to find a minimal set of axioms that is responsible for inconsistencies in an ontology was first proposed in Baader et al. (2005). In order to detect a set of problematic axioms, assertions are labeled and traced back, if a contradiction is found in a tableau expansion tree. In Schlobach and Cornet (2003), an advanced approach of this idea is presented by introducing the notion of a *minimal unsatisfiability-preserving sub-TBox* (MUPS): An axiom pinpointing service for $\mathcal{ALC}$ is proposed identifying the exact parts of axioms that are causing a contradiction. Several present approaches to ontology debugging are concerned with explanation services that are integrated into ontology developing tools. For example, Wang et al. (2005) present a service explaining unsatisfiability in OWL-DL ontologies by highlighting problematic axioms and giving natural language explanations of the conflict. In Haase and Stojanovic (2005), an approach to automatic ontology extraction is described. Every extracted axiom receives a confidence rating witnessing how frequent the axiom occurs in external sources.

The approaches sketched above either do not give solutions of how to fix the discovered contradictions or just propose to remove a problematic part of an axiom, although removed parts of axioms can result in a loss of information. Considering, for example, Example 1 again, if the concept $CanFly$ is removed from axiom 1, then the entailments $Bird \sqsubseteq CanMove$ and $Canary \sqsubseteq CanFly$ are lost.

In Fanizzi et al. (2004), inductive logic programming techniques are proposed to resolve inconsistencies. If a concept C is unsatisfiable, then the axiom defining C is replaced by a new axiom, constructed on the basis of positive assertions for C. The information previously defined in the ontology for C gets lost. Kalyanpur (2006) extends the OWL-DL tableau algorithm with a tracing technique to detect conflicting parts of axioms. It is suggested to rewrite axioms using frequent error patterns occurring in ontology modeling. Lam et al. (2006) revise the technique proposed in Baader and Hollunder (1995) and support ontology engineers in rewriting problematic axioms in $\mathcal{ALC}$: Besides the detection of conflicting parts of axioms, a concept is constructed, that replaces the problematic part of the chosen axiom. This approach keeps the entailment $Bird \sqsubseteq CanMove$, but not $Canary \sqsubseteq CanFly$ in Example 1. An approach to resolve overgeneralized concepts conflicting with exceptions is presented in Ovchinnikova and

Kühnberger (2006) for $\mathcal{ALE}$. Besides rewriting problematic axioms, a split of an overgeneralized concept C into a more general concept (not conflicting with exceptions) and a more specific one (capturing the original semantics of C) is proposed.

5 Proposed Approach

5.1 Tracing Clashes

In this section, we revise the tableau-based algorithm presented in Lam et al. (2006) for tracing clashes in unsatisfiable terminologies and rewriting problematic axioms. We adapt this tracing technique to our simple logic. The proposed algorithm detects the relevant parts of the axioms that are responsible for the contradiction.

Following Lam et al. (2006) suppose that a terminology $\mathcal{T}$ contains axioms $\{\alpha_1, ..., \alpha_n\}$, where α_i refers to an axiom $A_i \sqsubseteq C_i$ or $A_i \equiv C_i$ ($i \in \{1, ..., n\}$). In checking the satisfiability of a concept description C the tableau algorithm constructs a model of C represented by a tree $\mathbf{T}$. Each node x in this tree is labeled with a set $\mathcal{L}(x)$ containing elements of the form $(a : C, I, a' : C')$, where C and C' are concept descriptions, a and a' are individual names, and I is a set of axiom indices. An element $(a : C, I, a' : C')$ has the following intended meaning: individual a belongs to concept description C due to the application of an expansion rule on C' and I contains the indices of the axioms where $a : C$ originates from.

$\mathbf{T}$ is initialized with a node x and $\mathcal{L}(x) = \{(a : C, \varnothing, nil)\}$. The algorithm expands $\mathbf{T}$ according to the rules in Table 1[5]. Concept descriptions involved in the expansion are converted to negation normal form. The algorithm terminates if no more expansion rules can be applied to tree nodes. $\mathbf{T}$ contains a *clash* if an individual a belongs simultaneously to concept descriptions C and $\neg C$, i.e. $(a : C, -, -) \in \mathcal{L}(x)$ and $(a : \neg C, -, -) \in \mathcal{L}(x)$.[6]

A clash in an expansion tree does not always mean unsatisfiability. If an individual a in a model tree for a concept A belongs to a value restriction $\forall R.C$, where C is unsatisfiable, but a has no R-successors, then this value restriction does not cause unsatisfiability of A. On the other hand, with the advent of instances or subconcepts of A which have R-successors this value restriction invokes unsatisfiability (cf. Wang et al., 2005).

Definition 2 *Minimal clash-preserving sub-TBox (MCPS)*
Let A be a concept for which its model tree obtained relative to a terminology $\mathcal{T}$ contains clashes. A sub-TBox $\mathcal{T}' \subset \mathcal{T}$ is a MCPS of A, if a model tree for A towards $\mathcal{T}'$ contains clashes and a model tree of A towards every $\mathcal{T}'' \subset \mathcal{T}'$ contains no clashes.

The union of the axiom indices in I of all clash elements in the model tree of A corresponds to MUPS[7] of A in Schlobach and Cornet (2003) and constitutes MCPS of A in Lam et al. (2006). Given a specific clash (e_1, e_2), $\text{MCPS}_{(e_1,e_2)}(A)$ is similarly

[5] Tags are provided to avoid circularity in the expansion of concept definitions.
[6] "–" is a placeholder. It stands for any value.
[7] Concerning unsatisfiablitily of A.

Table 1: Tableau expansion rules.

Rule $\equiv+$	if $A_i \equiv C_i \in \mathcal{T}$, and $(a : A_i, I, a' : A')$ is not tagged, then $tag((a : A_i, I, a' : A'))$ and $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C_i, I \cup \{i\}, a : A_i)\}$
Rule $\equiv-$	if $A_i \equiv C_i \in \mathcal{T}$, and $(a : \neg A_i, I, a' : A')$ is not tagged, then $tag((a : \neg A_i, I, a' : A'))$ and $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : \neg C_i, I \cup \{i\}, a : \neg A_i)\}$
Rule $\sqsubseteq$	if $A_i \sqsubseteq C_i \in \mathcal{T}$, and $(a : A_i, I, a' : A')$ is not tagged, then $tag((a : A_i, I, a' : A'))$ and $\mathcal{L}(x) := \mathcal{L}(x) \cup \{(a : C_i, I \cup \{i\}, a : A_i)\}$
Rule $\forall$	if $(a : \forall R.C, I, a' : A') \in \mathcal{L}(x)$, and the above rules cannot be applied, then if there is $(b : D, J, a : \forall R.D) \in \mathcal{L}(x)$, then $\mathcal{L}(x) := \mathcal{L}(x) \cup \{b : C, I, a : \forall R.C\}$ else $\mathcal{L}(x) := \mathcal{L}(x) \cup \{b : C, I, a : \forall R.C\}$, where b is a new individual name

defined as in Definition 2 except for $\text{MCPS}_{(e_1,e_2)}(A)$ preserving only one clash (e_1, e_2), but not all clashes as $\text{MCPS}(A)$. To trace clashes we need to introduce the following definition.

Definition 3 *Trace*
Given an element $e = (a_0 : C_0, I_0, a_1 : C_1)$ *in a set* $\mathcal{L}(x)$*, the trace of* e *is a sequence of the form* $\langle (a_0 : C_0, I_0, a_1 : C_1), (a_1 : C_1, I_1, a_2 : C_2), \ldots, (a_{n-1} : C_{n-1}, I_{n-1}, a_n : C_n), (a_n : C_n, \varnothing, nil)\rangle$*, where* $I_{i-1} \subseteq I_i$ *for each* $i \in \{1, ..., n\}$ *and every element in the sequence belongs to* $\mathcal{L}(x)$*.*

Note that the expansion rules in Table 1 coincide with Lam et al. (2006), except for the Rule $\forall$, which obviously does not change crucial properties of the algorithm like complexity, decidability etc. Therefore, the properties of the original algorithm in Lam et al. (2006) are also relevant for our algorithm.

5.2 Types of Clashes

First of all, it is important to understand which solution for resolving clashes is appropriate from a pragmatic point of view. In order to achieve this, we return to our running examples. Concerning Example 1 it seems to be obvious that the axiom $Bird \sqsubseteq CanFly$ has to be modified, since this axiom contains overgeneralized knowledge. Simply deleting this axiom would result in the loss of the entailments $Bird \sqsubseteq CanMove$ and $Canary \sqsubseteq CanFly$, although both entailments do not contradict with the axiom $Penguin \sqsubseteq \neg CanFly$. A natural idea is to replace the problematic part of the overgeneralized definition of the concept $Bird$ (namely $CanFly$) with its least subsumer, that does not conflict with $Penguin$. In our example, the concept description $CanMove$ is precisely such a subsumer. Unfortunately, the simple replacement of $CanFly$ by $CanMove$ in Axiom 1 is not sufficient to preserve the entailment $Canary \sqsubseteq CanFly$.

We suggest therefore to introduce a new concept $FlyingBird$ that preserves the previous meaning of $Bird$ and subsumes its former subconcepts:

1. $Bird \sqsubseteq CanMove$
2. $CanFly \sqsubseteq CanMove$
3. $Canary \sqsubseteq FlyingBird$
4. $Penguin \sqsubseteq Bird$
5. $Penguin \sqsubseteq \neg CanFly$
6. $FlyingBird \sqsubseteq Bird$
7. $FlyingBird \sqsubseteq CanFly$

The situation is different for multiple overgeneralizations. A relevant solution for Example 2 is to replace the overgeneralized definitions $\forall likes.Icecream$ and $\forall likes.Chocolate$ with their least common subsumer $\forall likes.Sweetie$. The resulting axiom $Child \sqsubseteq \forall likes.Sweetie$ claims that children like only sweeties without specifying it:

1. $Child \sqsubseteq \forall likes.Sweetie$ 2. $Icecream \sqsubseteq Sweetie$ 3.$Chocolate \sqsubseteq Sweetie$
4. $Icecream \sqsubseteq \neg Chocolate$ 5. $Chocolate \sqsubseteq \neg Icecream$

The examples make clear that it is a non-trivial practical question of how multiple and single overgeneralizations can be distinguished. A single overgeneralization occurs, if some concept is too specifically defined and an exception contradicts with this definition. In the case of multiple overgeneralizations, two or more definitions of the same concept are too specific and conflict with each other. Unfortunately, it seems to be impossible to define this distinction purely logically, since this distinction is just a matter of human expert interpretation.

From a practical perspective it turns out that multiple overgeneralizations occur, if a concept is subsumed by two or more concepts that are explicitly defined as disjoint in the ontology (cf. Example 2). This case has a certain structural similarity to the polysemy problem, where an unsatisfiable concept is also subsumed by different disjoint concepts (cf. the tree example in Section 3). Practically, polysemy can be distinguished from multiple overgeneralizations by taking into account the level of abstraction of the disjoint concepts. In the case of polysemy, the disjoint superconcepts of the unsatisfiable concept usually occur in the upper structure of the taxonomy tree, whereas multiple overgeneralizations occur on lower levels of the taxonomy.

Definition 4 defines the abstraction level of concepts. The abstraction level of a concept towards a model tree is the number of steps in the shortest path from this concept to a most general (undefined) concept in the tableau extension procedure.

Definition 4 *Given a set $\mathcal{L}(x)$ that was obtained relative to a terminology $\mathcal{T}$ and an element $(a : C_1, I_1, a_0 : C_0) \in \mathcal{L}(x)$, the abstraction level $\mathbf{L}(C_1)$ is defined as the minimal cardinality of the sequences $\langle (a : C_1, I_1, a_0 : C_0), ..., (a : C_n, I_n, a : C_{n-1})\rangle$ where $\forall i \in \{1, ..., n\} : [(a : C_i, I_i, a_{i-1} : C_{i-i}) \in \mathcal{L}(x)$ and $I_i \subseteq I_{i-1}]$ and there is no concept D such that $C_n \sqsubseteq D \in \mathcal{T}$ or $C_n \equiv D \in \mathcal{T}$.*

Using Definition 4 it is possible to distinguish the two types of inconsistencies formally (cf. Definition 5): If a concept is subsumed by two other concepts that are defined to be disjoint[8] and the abstraction level of these concepts is higher than a user-defined

[8] For the sake of simplicity Definition 5 concerns multiple overgeneralization with only two concepts.

distinctive abstraction level, then this case is considered to be polysemous. If the abstraction level is below the user-defined abstraction level, then we are dealing with multiple overgeneralizations. Finally, if the clash is not produced by explicitly disjoint concepts, then we face the case of single overgeneralization.

Definition 5 *Given a clash* $(a : C, -, -), (a : \neg C, -, -)$ *from a set* $\mathcal{L}(x)$ *that was obtained relative to a terminology* $\mathcal{T}$ *and a distinctive abstraction level* l*, the following cases can be distinguished:*

- *If there exists a concept* D *such that* $(a : D, -, -), (a : \neg D, -, -) \in \mathcal{L}(x)$ *and* $D \sqsubseteq \neg C \in \mathcal{T}$ *and* $C \sqsubseteq \neg D \in \mathcal{T}$*, then*
  - *If* $\mathtt{max}(\mathbf{L}(C), \mathbf{L}(D)) \geq l$ *this clash is polysemous,*
  - *Else this clash is a multiple overgeneralization,*
- *Else this clash is a single overgeneralization.*

In the following subsection, we will discuss resolution aspects of the mentioned types of clashes.

5.3 Resolving Clashes

Unfortunately, it is impossible to resolve polysemy problems automatically without an appeal to external knowledge. After splitting the problematic concept (e.g. *Tree* in the example of Section 3) into two concepts with different names (e.g. *TreeStructure* and *TreePlant*) it is necessary to find out which one of the definitions and subconcepts of the original concept refers to which of the new concepts. This can be done either by the ontology engineer or with the help of additional knowledge about the usage context of this concept in external resources. Since this paper is concerned with logical aspects of ontology adaptation only, we do not consider this problem here.

As already mentioned above, multiple overgeneralizations can be repaired by replacing conflicting definitions with their least common subsumer. In order to find a least common subsumer, we need to calculate subsumers for concepts. Fact 1 characterizes subsumers computationally.

Fact 1 *Given a set* $\mathcal{L}(x)$ *obtained relative to a terminology* $\mathcal{T}$ *and concept* C *such that* $(a : C, -, -) \in \mathcal{L}(x)$*, a concept* C' *is a subsumer of* C *towards* $\mathcal{T}$ *if*

- $\exists e = (a : C', -, -) : (a : C, -, -) \in Trace(e)$ *or*
- $\exists e = (a : \forall R.D, -, -) : (a : C, -, -) \in Trace(e)$ *and* $C' \doteq \forall R.D'$ *such that* D' *is atomic and a subsumer of* D*.*

If C *is satisfiable towards* $\mathcal{T}$*, then the other direction of the implication does also hold.*

Fact 1[9] claims that a concept C' is a subsumer of a concept C, if it was added to a node a in the tableau expansion procedure after C or if C is subsumed by a relational

[9] The proofs of Fact 1 and further facts below are not presented in detail due to space limitations. We will rather provide sketches of proof ideas.

restriction $\forall R.D$ and C' is a relational restriction on R with a scope D' subsuming D. General axioms with a complex concept definition on the right side cannot occur in our restricted logic. Therefore if C is satisfiable, then no inferred subsumption that is not explicitly expressed in the TBox is possible. If C is unsatisfiable, then it is subsumed by any concept. As the reader will see hereinafter, we are interested only in cases where C is satisfiable.

According to Fact 2, given a set $\mathcal{L}(x)$, a lcs for two satisfiable concepts C_1 and C_2 occurring in $\mathcal{L}(x)$ can be characterized as a minimal concept subsuming both, C_1 and C_2 towards $\mathcal{T}$.

Fact 2 *Given a set $\mathcal{L}(x)$ obtained relative to a terminology $\mathcal{T}$ and concepts C_1 and C_2 satisfiable towards $\mathcal{T}$, such that $(a : C_1, -, -) \in \mathcal{L}(x), (a : C_2, -, -) \in \mathcal{L}(x)$ a concept L is a least common subsumer for C_1 and C_2 towards $\mathcal{T}$ if and only if the following two conditions hold:*

1. $L \in subsumers_{\mathcal{T}}(C_1) \cap subsumers_{\mathcal{T}}(C_2)$.
2. *For every concept L' that satisfies condition 1 and is not equal to L: $\mathcal{T} \not\models L' \sqsubseteq L$.*

Fact 1 ensures that L satisfies condition 1 of Definition 1 (recall that the set $subsumers_{\mathcal{T}}(C)$ is the exhaustive set of concept descriptions subsuming C towards $\mathcal{T}$ provided C is satisfiable towards $\mathcal{T}$). Obviously, L satisfies also condition 2 of Definition 1. Thus, in order to resolve a clash produced in a model tree of a concept A by two overgeneralized definitions C_1 and C_2, it is sufficient to delete axioms $A \sqsubseteq C_1$ and $A \sqsubseteq C_2$ from the terminology and add axioms from the set $\{A \sqsubseteq L \mid L \text{ is a } lcs(C_1, C_2)\}$. If concepts C_1 and C_2 themselves are unsatisfiable, then they should be repaired before A. Compare Section 7 for more remarks concerning this issue.

Now we examine the problem of resolving the case of single overgeneralization. Lam et al. (2006) shows that removing any of the axioms appearing in the clash traces is sufficient to resolve the clash. An important practical question concerns the choice of the axiom to be removed or modified. In the literature, a lot of ranking criteria were suggested for this task on ontology debugging (Schlobach and Cornet, 2003; Kalyanpur, 2006; Lam et al., 2006; Haase and Stojanovic, 2005):

- *Arity* of an axiom α denotes in how many clashes α is involved. The higher the arity is, the lower is the rank of α.
- *Semantic impact* of α denotes how many entailments are lost if α is removed. Axioms with a high semantic impact are ranked higher.
- *Syntactic relevance* denotes how often concept and role names occurring in an axiom α are used in other axioms in the ontology. Axioms containing elements that are frequently occurring in the ontology are ranked higher.
- *Manual ranking* of α can be provided by the ontology engineer.

- *Frequency ranking* of α is used in approaches to semi-automatic ontology extraction and denotes how often concepts and roles in α occur in external data sources.

In this paper, we do not discuss ranking strategies and suppose that one of these strategies has been applied and the problematic axiom to be removed or rewritten has been chosen. Assume a concept description C is chosen to be removed from an axiom α, in order to resolve a clash in a model tree for a concept A. The next questions is: How can we find an appropriate concept description C' that can resolve the clash by replacing C? We are looking for a replacement that resolves the clash, does not cause new clashes or entailments, and preserves as many entailments implied by $\mathcal{T}$ as possible. Definition 6 defines such a replacement.

Definition 6 *Minimal nonconflicting substitute (MNS)*
Assume the following is given: a terminology $\mathcal{T}$, a clash $e_1 = (a : X, -, -), e_2 = (a : \neg X, -, -)$ in the model tree for a concept A, a concept C satisfiable towards $\mathcal{T}$[10] *that is chosen to be removed from an axiom α_i ($\alpha_i \in \mathcal{T}$ and $\exists j \in \{1, 2\} : (a : C, \{i, -\}, -) \in Trace(e_j)$), and $\mathcal{T}'' := \mathcal{T} \setminus \{\alpha_i\}$. Let an axiom α' be obtained from α_i by replacing C with a concept C' and $\mathcal{T}' := \mathcal{T} \setminus \{\alpha_i\} \cup \{\alpha'\}$. C' is a minimal nonconflicting substitute (MNS) of C if the following conditions hold:*

1. *A model tree for A towards $\mathcal{T}'$ contains the same number of clashes as a model tree for A towards $\mathcal{T}''$.*
2. *If $C' \neq \top$, then there exists an entailment β, such that $\mathcal{T} \models \beta$, $\mathcal{T}'' \not\models \beta$, and $\mathcal{T}' \models \beta$.*
3. *There exists no entailment β such that $\mathcal{T} \not\models \beta$ and $\mathcal{T}' \models \beta$.*
4. *There exists no concept description C'' with the same properties of C', such that C'' preserves more entailments from $\mathcal{T}$.*

Condition (1) guarantees that MNS resolves the clash (e_1, e_2) in which α_i and C are involved and does not introduce new clashes. Due to condition (2) MNS preserves at least one entailment from $\mathcal{T}$ that would be lost with the removal of C. Condition (3) excludes new entailments that are not implied by $\mathcal{T}$ and condition (4) guarantees that MNS preserves as much information as possible.

Fact 3 *Given a clash $e_1 = (a : X, -, -), e_2 = (a : \neg X, -, -)$ obtained from a set $\mathcal{L}(x)$ relative to a terminology $\mathcal{T}$ and a concept C satisfiable towards $\mathcal{T}$ that is chosen to be removed from an axiom α_i ($\alpha_i \in \mathcal{T}$ and $\exists j \in \{1, 2\} : (a : C, \{i, -\}, -) \in Trace(e_j)$), a concept C' is a MNS of C if and only if the following conditions hold:*

1. *C' subsumes C towards $\mathcal{T}$.*

[10] Notice again: if C is unsatisfiable, then it should be repaired before A. Compare Section 7 for more information.

2. *If C is subsumed by X or $\neg X$ towards $\mathcal{T}$, then C' is not subsumed by X or $\neg X$ towards $\mathcal{T}$.*

3. *C' preserves at least one entailment from $\mathcal{T}$ or $C' := \top$.*

4. *There exists no concept description $C'' \neq C'$ with the same properties, such that C' subsumes C'' towards $\mathcal{T}$.*

Due to conditions (1) and (2) C' satisfies conditions (1) and (3) of Definition 6 (since C' is not subsumed by a conflicting definition, it does not reconstruct the clash). Condition (3) guarantees that C' satisfies condition (2) of Definition 6. Finally condition (4) corresponds to (4) in Definition 6.

We reconsider Example 1. The model tree for the concept $Penguin$ in this example consists of the following elements:

$$\begin{aligned}\mathcal{L}(x) \quad = \quad & \{(a : Penguin, \varnothing, nil), (a : Bird, \{4\}, a : Penguin),\\ & (a : \neg CanFly, \{5\}, a : Penguin), (a : CanFly, \{4, 1\}, a : Bird),\\ & (a : CanMove, \{4, 1, 2\}, a : CanFly)\}\end{aligned}$$

Thus, the set of problematic axioms is $\{1, 4, 5\}$. Suppose the concept $CanFly$ is chosen to be removed from axiom 1. According to Fact 3 $CanMove$ is MNS of $CanFly$. If $CanFly$ is replaced by $CanMove$ in Axiom 1, then the entailments of the form $X \sqsubseteq CanFly$, where X is a subconcept of $Bird$ (for example, $Canary$), would be lost. Such situations are undesirable, because the clash $(CanFly, \neg CanFly)$ concerns only the conflict between the overgeneralized concept $Bird$ and the exception $Penguin$. In order to keep the entailments, we suggest to introduce a new concept $FlyingBird$ to the terminology which will capture the original meaning of $Bird$ (cf. Section 5.2).

6 Adaptation Algorithm

If a new axiom α is added to a terminology $\mathcal{T}$, then the proposed algorithm constructs model trees for every atomic concept X that is defined in $\mathcal{T}$. Model trees containing clashes are used for ontology repairing. Using Definition 5 the algorithm distinguishes between inconsistency types. Since polysemy can not be repaired without external knowledge, this problem is only reported to the user. Multiple overgeneralizations are repaired by replacing the conflicting definitions with their least common subsumer.

With respect to single overgeneralization, for every clash (e', e'') a concept description F from the trace of some clash element $e \in \{e', e''\}$ is chosen to be rewritten in an axiom β ($\beta \doteq E \sqsubseteq F$ or $\beta \doteq E \equiv F$) according to a given ranking (see section 5.3). F is replaced with its minimal nonconflicting substitutes in β. A new concept E^{new} is introduced to capture the original semantics of E. The name E^{new} is constructed automatically from the original name E and the problematic concept F. The set T consists of elements from $\mathcal{L}(x)$ that are contained in the trace of the element e between the unsatisfiable concept X and the rewritten concept E.

The **split&replace** procedure splits atomic concepts from $\mathcal{L}(x)$ that are involved in the clash above. Concepts appearing in the trace of e earlier are split first. If a definition of B_1 was rewritten and B_1 was split into B_1^{new} and B_1 and a concept B_2 is going to be split immediately after B_1, then B_1 in the definition of a new concept name B_2^{new} is replaced with B_1^{new}.

Algorithm Adapt a satisfiable terminology $\mathcal{T}$ to a new axiom α,
given a distinctive abstraction level l

$\alpha \doteq A \sqsubseteq B$ or $\alpha \doteq A \equiv B$, add α to $\mathcal{T}$
for all axioms $\alpha' \doteq X \sqsubseteq Y$ or $\alpha' \doteq X \equiv Y$, $\alpha' \in \mathcal{T}$
if X is unsatisfiable towards $\mathcal{T}$ **then**
  for all clashes (e', e'') in the sets $\mathcal{L}(x)$ of the model tree for X where
    $e' = (a : C, -, -), e'' = (a : \neg C, -, -)$
    if $\exists D : (a : D, -, -), (a : \neg D, -, -) \in \mathcal{L}(x)$ and $\{D \sqsubseteq \neg C, C \sqsubseteq \neg D\} \subset \mathcal{T}$
    then if $\mathtt{max}(\mathbf{L}(C), \mathbf{L}(D)) \geq l$ **then** report polysemy
      else remove D_1, D_2 where
        $D_{i \in \{1,2\}}$ are last but one elements in the traces of e', e''
        for all $lcs(D_1, D_2)$ add $A \sqsubseteq lcs(D_1, D_2)$ to $\mathcal{T}$ **end for**
    else
      choose an $\beta \in \mathcal{T}$ ($\beta \doteq E \sqsubseteq F$ or $\beta \doteq E \equiv F$) such that
      $\exists e \in \{e', e''\}:$ $(b : F, -, -) \in Trace(e)$ to be rewritten acc. to *ranking*
      remove β from $\mathcal{T}$
      for all MNS(F)
        β^{new} is obtained from β by replacement of F with MNS(F)
        add β^{new} to $\mathcal{T}$
      end for
      add $E^{new} \sqsubseteq E, E^{new} \sqsubseteq F$ to $\mathcal{T}$
      let T be a subsequence of $Trace(e)$
      between the elements $(a : E, -, -)$ and $(b : X, -, -)$ (not inclusive)
      split&replace(E, E^{new}, T)
  end for

Subroutine **split&replace**(A, A^{new}, T)
$(b : B', -, a : B)$ is the **next** element of T and B is atomic
B'' is obtained by replacing A with with A^{new} in B'
if $B \sqsubseteq B' \in \mathcal{T}$ **then** add $B^{new} \sqsubseteq B''$ to $\mathcal{T}$
else add $B^{new} \equiv B''$ to $\mathcal{T}$
for all $\gamma \in \mathcal{T}$ such that γ is not the next axiom in T
    replace B with B^{new} in the right part of γ
end for
split&replace(B, B^{new}, T)

Example 3 shows the application of the algorithm. The concept $Transport$ in axiom

2 is chosen to be rewritten. It is easy to see that the proposed algorithm extends the semantics of the "split" concepts, whereas the semantics of other concepts remains unchanged. New concept names ($TransportAirplane, AviatesTransportAirplanePilot$) are constructed automatically.

Example 3

Original terminology:

1. $Pilot \sqsubseteq \forall aviates.Airplane$
2. $Airplane \sqsubseteq Transport$
3. $PassengerPlane \sqsubseteq Airplane$
4. $FighterPilot \sqsubseteq Pilot$
5. $FighterPilot \sqsubseteq \forall aviates.FightingMachine$
6. $FightingMachine \sqsubseteq \neg Transport$

Changed terminology:

1. $Pilot \sqsubseteq \forall aviates.Airplane$
2. $PassengerPlane \sqsubseteq TransportPlane$
3. $FighterPilot \sqsubseteq Pilot$
4. $FighterPilot \sqsubseteq \forall aviates.\neg FightingMachine$
5. $FightingMachine \sqsubseteq \neg Transport$
6. $TransportAirplane \sqsubseteq Airplane$
7. $TransportAirplane \sqsubseteq Transport$
8. $AviatesTransportAirplanePilot \sqsubseteq Pilot$
9. $AviatesTrasportAirplanePilot \sqsubseteq \forall aviates.TrasportAirplane$

7 Root and Derived Concepts

It is easy to verify that the result of the application of our algorithm is dependent on the order the concepts are input into the debugger. If axioms are added one by one, then nothing in the procedure needs to be changed. But if a set of axioms is added to the ontology, then it is interesting to see whether it is reasonable to reorder axioms in this set. Unsatisfiable concepts can be divided into two classes:

1. *Root concepts* are atomic concepts for which a clash found in their definitions does not depend on a clash of another atomic concept in the ontology.

2. *Derived concepts* are atomic concepts for which a clash found in their definitions either directly (via explicit assertions) or indirectly (via inferences) depends on a clash of another atomic concept (Kalyanpur, 2006).

In order to debug a derived concept, it is enough to debug corresponding root concepts. Thus, it is reasonable to debug only the root concepts. The technique of distinguishing between root and derived concepts was proposed in Kalyanpur (2006). We integrate this technique into our approach.

Definition 7 *An atomic concept A is* **derived** *from an atomic concept A' if there exists a clash (e_1, e_2) in the model tree of A such that $MCPS_{(e_1,e_2)}(A')$ is a subset of*

$MCPS_{(e_1,e_2)}(A)$. If there is no concept A' from which A is derived, then A is a **root** *concept.*

Fact 4 shows how to find dependencies between problematic concepts.

Fact 4 *Given a clash (e_1, e_2) obtained from a set $\mathcal{L}(x)$ for a concept A, A is derived from a concept A' (towards this clash) if and only if $\exists(b : A', -, -) \in Trace_{\mathcal{L}(x)}(e_1) : (b : A', -, -) \in Trace_{\mathcal{L}(x)}(e_2)$.*

$\text{MCPS}_{(e_1,e_2)}(A')$ is a subset of $\text{MCPS}_{(e_1,e_2)}(A)$ if and only if the model tree for A' is a subset of the model tree for A and both of these trees contain the clash (e_1, e_2) (modulo differences in the axiom indices set). Obviously, A' is in the traces of the clash elements in its own model tree. Therefore, if A is derived from A' towards (e_1, e_2), then A' is in the traces of e_1 and e_2 in $\mathcal{L}(x)$. On the other hand, if A' occurs in the traces of e_1 and e_2, then the model tree of A' contains this clash and is a subset of $\mathcal{L}(x)$.

Using Fact 4 it is possible to construct a directed graph with atomic concepts as nodes and arrows denoting derivation. It can happen that two concepts are simultaneously derived from each other (for example $\{A \sqsubseteq D,\ A \sqsubseteq \neg D,\ B \sqsubseteq D,\ B \sqsubseteq \neg D,\ A \equiv B\}$). In this case, it is necessary to debug both of the derived concepts.

8 Conclusion and Future Work

In this paper, we presented an approach for dynamically resolving conflicts appearing in automatic ontology learning. This approach is an integration of ideas proposed in Lam et al. (2006) and Ovchinnikova and Kühnberger (2006) extended for the subset of description logics used in practically relevant systems for ontology learning (Haase and Stojanovic, 2005). Our algorithm detects problematic axioms that cause a contradiction, distinguishes between different types of logical inconsistencies and automatically repairs the terminology. This approach is knowledge preserving in the sense that it keeps as many entailments implied by the original terminology as possible.

In Ovchinnikova et al. (2007), a prototypical implementation of the idea of splitting overgeneralized concepts in $\mathcal{ALE}$-DL was discussed. This implementation was tested on the famous wine-ontology[11] that was automatically extended with new classes extracted from text corpora with the help of the Text2Onto[12] tool. Several cases of overgeneralization were detected and correctly resolved[13].

In the near future, we plan to test the prototype implementation of the proposed algorithm on existing real-life ontologies. It is of particular interest to see to what extent statistical information about the distribution and co-occurrence of concepts in texts can help to improve the adaptation procedure for making it more adequate to human intuition.

[11] `http://www.w3.org/TR/owl-guide/wine.owl`

[12] `http://ontoware.org/projects/text2onto/`

[13] For example, the class *LateHarvest* originally defined to be a sweet wine was claimed to be overgeneralized after an exception *RieslingSpaetlese* which was defined to be a late harvest wine and a dry wine appeared.

Acknowledgments

This research was partially supported by grant MO 386/3-4 of the German Research Foundation (DFG).

References

Baader, F., Calvanese, D., McGuinness, D. L., Nardi, D., and Patel-Schneider, P. F., editors (2003). *The Description Logic Handbook: Theory, Implementation, and Applications*. Cambridge University Press, New York.

Baader, F. and Hollunder, B. (1995). Embedding Defaults into Terminological Representation Systems. *J. Automated Reasoning*, 14:149–180.

Baader, F. and Küsters, R. (2006). Non-standard Inferences in Description Logics: The Story So Far. In Gabbay, D. M., Goncharov, S. S., and Zakharyaschev, M., editors, *Mathematical Problems from Applied Logic I. Logics for the XXIst Century*, volume 4 of *International Mathematical Series*, pages 1–75. Springer.

Baader, F., Lutz, C., Milicic, M., Sattler, U., and Wolter, F. (2005). Integrating Description Logics and Action Formalisms: First Results. In *Proceedings of the 2005 International Workshop on Description Logics (DL2005)*, CEUR-WS.

Fanizzi, N., Ferilli, S., Iannone, L., Palmisano, I., and Semeraro, G. (2004). Downward Refinement in the $\mathcal{ALN}$ Description Logic. In *HIS '04: Proc. of the Fourth International Conference on Hybrid Intelligent Systems (HIS'04)*, pages 68–73, Washington, DC, USA. IEEE Computer Society.

Haase, P. and Stojanovic, L. (2005). Consistent evolution of owl ontologies. In *Proc. of the Second European Semantic Web Conference*, pages 182–197.

Kalyanpur, A. (2006). *Debugging and Repair of OWL Ontologies. Ph.D. Dissertation.* University of Maryland College Park.

Lam, S. C., Pan, J. Z., Sleeman, D. H., and Vasconcelos, W. W. (2006). A Fine-Grained Approach to Resolving Unsatisfiable Ontologies. In *Web Intelligence*, pages 428–434.

Ovchinnikova, E. and Kühnberger, K.-U. (2006). Adaptive $\mathcal{ALE}$-Tbox for Extending Terminological Knowledge. In *19th Australian Joint Conference on ArtificialIntelligence*, pages 1111–1115.

Ovchinnikova, E., Wandmacher, T., and Kühnberger, K.-U. (2007). Solving Terminological Inconsistency Problems in Ontology Design. *International Journal of Interoperability in Business Information Systems (IBIS)*, 2(1):65–80.

Perez, G. A. and Mancho, M. D. (2003). A Survey of Ontology Learning Methods and Techniques. OntoWeb Delieverable 1.5.

Schlobach, S. and Cornet, R. (2003). Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies. In *IJCAI*, pages 355–362.

Wang, H., Horridge, M., Rector, A. L., Drummond, N., and Seidenberg, J. (2005). Debugging OWL-DL Ontologies: A Heuristic Approach. In *International Semantic Web Conference*, pages 745–757.