Black Box Debugging of Unsatisfiable Classes

Aditya Kalyanpur and **Bijan Parsia** and **Evren Sirin**
aditya@cs.umd.edu, bparsia@isr.umd.edu, evren@cs.umd.edu
Maryland Information and Network Dynamics Laboratory,
University of Maryland,
College Park, MD 20740, USA

Abstract

In this paper, we explore *black box* techniques for debugging unsatisfiable concepts expressed in the Web Ontology Language (OWL), that is, techniques which use the reasoner solely to determine which concepts are unsatisfiable, but use the asserted structure of the ontology to help isolate the source of the problems. We limit ourselves to two key debugging tasks, detecting dependencies between unsatisfiable concepts and pinpointing and explaining problematic axioms in the root concepts.

1 Introduction

In [3], we investigated better support for debugging unsatisfiable concepts using both glass box (wherein the reasoner is modified to return explanations of the unsatisfiability) and black box (wherein the only inference service is satisfiability checking) techniques. In this paper, we extend our investigation of black box techniques which have two advantages over glass box ones: reasoner independence (you do not need a specialized, explanation generating reasoner) and avoiding the performance penalty of glass box techniques.

Furthermore, we focus our attention on two key tasks: detecting dependencies between unsatisfiable classes and presenting problematic axioms for a set of core classes.

2 Dependency between Unsatisfiable Classes

We distinguish between *root* and *derived* unsatisfiable classes as follows: a *root* class is an unsatisfiable class in which a clash or contradiction found in the class definition (axioms) does not **depend on the unsatisfiability** of another class in the ontology. Whereas, a *derived* class is an unsatisfiable class in which a clash or contradiction found in a class definition either directly (via explicit assertions) or indirectly (via inferences) **depends on the unsatisfiability** of another class (we refer to it as the *parent* unsatisfiable class).

We have devised an algorithm to separate root from derived unsatisfiable classes in an ontology as provided by a reasoner. The algorithm consists of two parts: *asserted* dependency detection and *inferred* dependency detection.

2.1 Asserted Dependency Detection: Structural Tracing

This phase is used to detect dependencies between unsatisfiable classes by analyzing the asserted structure of the ontology. We present the basic cases of the tracing approach: Class A is a derived unsatisfiability if:

1. A is equivalentTo/subClassOf an intersection set, *any* of whose elements are unsatisfiable, i.e., $A = (B \cap C.. \cap D)$, and one of B,C..D is unsatisfiable (any such unsatisfiable class becomes its parent)
2. A is equivalentTo/subClassOf a union set, *all* of whose elements are unsatisfiable, i.e., $A = (B \cup C.. \cup D)$, and all B,C..D are unsatisfiable (all such unsatisfiable classes become its parents)
3. A has an existential ($\exists$) property restriction on an unsatisfiable class, i.e., $A = \exists(p, B)$ and B is unsatisfiable (B becomes its parent)
4. A entails a ($\geq$ 1) cardinality restriction on a property-chain, and the universal ($\forall$) value restriction on that chain is not satisfied (object/value of property chain becomes its parent)
5. A entails a ($\geq$ 1) cardinality restriction on a property, and the domain of the property is unsatisfiable, i.e., $A \sqsubseteq (\geq 1p), domain(p) = B$, and B is unsatisfiable, making it the parent of A (similar $domain$ check has to be made for every ancestor property of p)
6. A entails a ($\geq$ 1) cardinality restriction on an object property, and the range of its inverse is unsatisfiable, i.e., $A \sqsubseteq (\geq 1p), range(p^-) = B$, and B is unsatisfiable, making it the parent of A (similar $range$ check has to be made for every ancestor property of p^-)

2.2 Inferred Dependency Detection

The problem with detecting hidden dependencies in an incoherent KB (TBox) is that the unsatisfiability masks some useful subsumption relationships in the TBox. Hence, given a TBox with unsatisfiable concepts, we consider a *subsumption safe* transformation as one which modifies the TBox by trying to get rid of all inconsistencies while attempting to preserve the intended subsumption hierarchy as much as possible. After approximating the KB, we can use the reasoner to classify the (relevant part of) KB to detect hidden dependencies between concepts not caught in structural tracing. Moreover, if the concepts turn out to be satisfiable, and hidden dependencies are revealed, we can use the transformations performed to pinpoint axioms which cause incoherence in the core (parent) unsatisfiable classes.

3 Explaining Sets of Support for Root Classes

Having identified the core set of unsatisfiable classes in the ontology, we now focus on pinpointing the axioms leading to the contradiction of such classes. Moreover, since our emphasis is on explaining the circumstances leading to the contradiction, we present the axioms in a particular order to aid understanding of the problem better.

We illustrate this method with an example. Consider the following TBox containing 8 axioms:

(1) $A \sqsubseteq B \sqcap D$

(2) $B \sqsubseteq \exists p^{-}.H$

(3) $C \sqsubseteq \exists p.\neg F$

(4) $D \sqsubseteq (\leq 1)p$

(5) $E \sqsubseteq \forall p.G \sqcap H$

(6) $G \sqsubseteq F$

(7) $domain(p) = E$

(8) $range(p) = C$

The TBox (T) has two interesting properties:

1. Only class A is unsatisfiable
2. A is satisfiable in any sub-TBox, $T' \subset T$. Thus, the above set of axioms is the minimal TBox that captures the contradiction in A, or in other words, $MUPS(A)$ as defined in [4].

The above example shows that even identifying the $MUPS$ of a class, which itself is a non-trivial task, does not make it obvious why the class is unsatisfiable. Thus we must take into account the interaction among the axioms, which in turn can be used to explain the cause for the unsatisfiability.

At this point, we revert back to a glass box technique described in our earlier work [3] in which we obtain the sets of support for an unsatisfiable class (above 8 axioms in this case) and also display *clash information* of the form: 'Class A is unsatisfiable because it is a subclass of both, classes X and Y, which in the above case amounts to the following explanation: $A \sqsubseteq \exists p.\neg F \sqcap \forall p.F$. Again, while this is a step in the right direction from a debugging point of view, it still does not tell us *how* the specific subsumption relation shown above came about. Now using this feature as a starting point, we present a black box heuristic approach to explain the cause of the contradiction in two steps:

- **Step 1.** For each satisfiable element of the pair (X,Y) that is responsible for the clash, we obtain a set of all descendant subclasses (using the reasoner). In this case, we get the following two subsumption paths:
  $B \sqsubseteq C \sqsubseteq \exists p.\neg F$
  $D \sqsubseteq E \sqsubseteq \forall p.F$
  We can then prefix the relation $A \sqsubseteq ..$ before each path to complete the trace diagram as follows:
  $A \sqsubseteq B \sqsubseteq C \sqsubseteq \exists p.\neg F$
  $A \sqsubseteq D \sqsubseteq E \sqsubseteq \forall p.F$
  Already, such a trace is useful for leading the ontology debugger to the contradiction.

- **Step 2.** We overlay the subsumption trace diagram with axioms, i.e for each subsumption link between a pair of classes, we display the axioms that entail the subsumption relation. For this, we use our earlier glass box approach for sets of support (i.e. axioms responsible for unsatisfiable class) to capture axioms responsible for subsumption (since subsumption is reduced to consistency-check in tableaux calculus).

  The final subsumption trace with axiom sets overlaid looks like this ($\rightarrow$ represent subclass relationships):
  $A \overset{(1)}{\rightarrow} B \overset{(2,8)}{\rightarrow} C \overset{(3)}{\rightarrow} \exists p.\neg F$
  $A \overset{(1)}{\rightarrow} D \overset{(4,7)}{\rightarrow} E \overset{(5,6)}{\rightarrow} \forall p.F$
  Such a trace makes the cause for the contradiction in the unsatisfiable class clearer by systematically leading the ontology debugger to the source of the problem, at each point revealing only the relevant axioms which further it along the trace.

4 Conclusion & Future Work

In the black box debugging techniques described in this paper, we primarily focus on separating the root from the derived unsatisfiable classes allowing the modeler to focus solely on the problematic parts of the ontology. As a subsequent step, we are working on explaining unsatisfiability for the root classes from the sets of support axioms.

Evaluation performed on the Tambis OWL ontology, which contains 144 unsatisfiable classes, has given us promising preliminary results – structural tracing discovered 111 derived unsatisfiable classes, and after inferred dependency detection, only 2 roots remained.

As future work, we plan on extending structural tracing to capture more derived dependencies, and on optimizing the inferred dependency detection algorithm. Finally, we are working on heuristics for black box determination of MUPS.

References

[1] Aditya Kalyanpur, Bijan Parsia, and James Hendler. A tool for working with web ontologies. *International Journal on Semantic Web and Information Systems*, 1(1), Jan - March 2005.

[2] Bijan Parsia and Evren Sirin. Pellet: An owl dl reasoner (poster). In *Third International Semantic Web Conference (ISWC2004), Hiroshima, Japan*, Nov 2004.

[3] Bijan Parsia, Evren Sirin, and Aditya Kalyanpur. Debugging owl ontologies. In *Proceedings of the 14th International World Wide Web Conference (WWW2005)*, Chiba, Japan, May 2005.

[4] S. Schlobach and R. Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proceedings of the eighteenth International Joint*

Conference on Artificial Intelligence, IJCAI'03. Morgan Kaufmann, 2003.

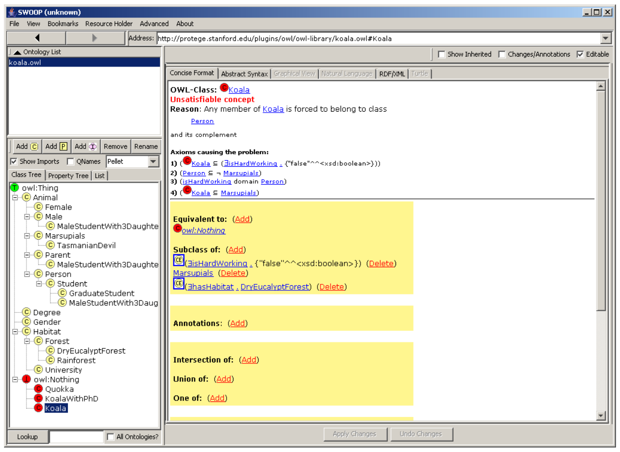
5 Demo using Swoop and Pellet

For the purpose of evaluation, we implemented the black-box techniques in the Swoop OWL ontology editor [1], used the default DL Tableaux Reasoner, Pellet [2], and carried out the analysis and debugging of various ontologies.

We intend to demo the debugging of OWL Ontologies using Swoop/Pellet. The following features have been integrated in the tool to support exploration and debugging of the ontology:

- Different rendering styles, formats, and icons are used to highlight key entities and relationships that that are likely to be helpful to debugging process. For example, all *inferred* relationships (axioms) in a specific entity definition are italicized and are obviously not editable directly. On a similar note, in the case of multiple ontologies, i.e., when one ontology imports another, all *imported* axioms in a particular entity definition are italicized as well. Highlighting them helps the modeler differentiate between explicit assertions in a single context and the net assertions (explicit plus implied) in a larger context (using imports), and can also reveal unintended semantics.
- All unsatisfiable named classes, and even class expressions, are marked with red icons whenever rendered — a useful pointer for identifying dependencies between inconsistencies. The hypertextual navigation feature of Swoop allows the user to follow these dependencies easily, and reach the root cause of the inconsistency, e.g., the class which is independently inconsistent in its definition (i.e., no red icons in its definition). In this manner, the UI guides the user in locating and understanding bugs in the ontology by narrowing them down to their exact source.
- Class expressions themselves can be rendered as regular classes, displaying information such as sub/super classes of a particular expression. This sort of ad hoc "on-demand" querying (e.g., find all subclasses of a specific query expression) helps reveal otherwise hidden dependencies.
- Swoop has a feature known as *Show References* highlights the usage of an OWL entity (concept/property/individual) by listing all references of that entity in local or external ontological definitions.
- Currently, glass box techniques in Swoop are used to support two forms of debugging of unsatisfiable concepts:
  1. Present the *Clash* information: root cause of the contradiction
  2. Determine the minimal *Sets of Support*: the relevant axioms in the ontology that are responsible for the clash
- Black box techniques are used to distinguish between root and derived unsatisfiable classes.

Following are some screenshots of the *debug* mode of Swoop in action.



Address: http://protege.stanford.edu/plugins/owl/owl-library/koala.owl#Koala

OWL-Class: Koala
Unsatisfiable concept
Reason: Any member of Koala is forced to belong to class
Person
and its complement

Axioms causing the problem:
1) (Koala ⊆ (∃isHardWorking . {"false"^^<xsd:boolean>}))
2) (Person ⊆ ¬ Marsupials)
3) (isHardWorking domain Person)
4) (Koala ⊆ Marsupials)

Equivalent to: (Add)
owl:Nothing

Subclass of: (Add)
(∃isHardWorking . {"false"^^<xsd:boolean>}) (Delete)
Marsupials (Delete)
(∃hasHabitat . DryEucalyptForest) (Delete)

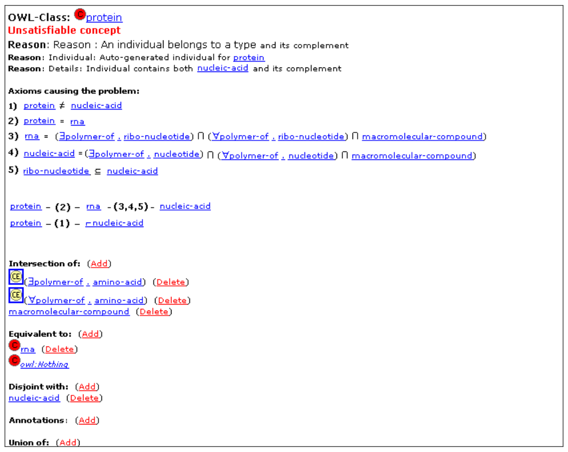
Annotations: (Add)

Intersection of: (Add)

Union of: (Add)

One of: (Add)

Figure 1: **Displaying the Sets of Support Axioms** The set of axioms that support the inconsistency of a concept is displayed in glass box debug mode.


OWL-Class: protein
Unsatisfiable concept
Reason: Reason : An individual belongs to a type and its complement
Reason: Individual: Auto-generated individual for protein
Reason: Details: Individual contains both nucleic-acid and its complement

Axioms causing the problem:
1) protein ≠ nucleic-acid
2) protein = rna
3) rna = (∃polymer-of . ribo-nucleotide) ⊓ (∀polymer-of . ribo-nucleotide) ⊓ macromolecular-compound)
4) nucleic-acid =(∃polymer-of . nucleotide) ⊓ (∀polymer-of . nucleotide) ⊓ macromolecular-compound)
5) ribo-nucleotide ⊆ nucleic-acid

protein – (2) – rna -(3,4,5)- nucleic-acid
protein – (1) – ¬nucleic-acid

Intersection of: (Add)
(∃polymer-of . amino-acid) (Delete)
(∀polymer-of . amino-acid) (Delete)
macromolecular-compound (Delete)

Equivalent to: (Add)
rna (Delete)
owl:Nothing

Disjoint with: (Add)
nucleic-acid (Delete)

Annotations: (Add)

Union of: (Add)

Figure 2: **Explaining Unsatisfiability**: Presenting axioms leading to contradiction for the unsatisfiable class `Protein` in the Tambis Ontology. `Protein` becomes a top-level *parent* class in the second debugging iteration.