An Approach to Ontology Debugging with Truth Maintenance Systems*

Hai H. Nguyen

University of Nottingham,UK
hhn@cs.nott.ac.uk

Abstract. This work presents a proposal to use Truth Maintenance Systems for ontology debugging tasks. It show the mapping between the axiom pinpointing problem and TMSs' operations. In particular, the labelling techniques in Assumption-based Truth Maintenance Systems are employed to pinpoint problematic axioms in an incoherent ontology.

1 Introduction

An ontology is a description of a particular domain in terms of its concepts and relationships. Ontologies can be used by 'intelligent agents' to model their environment and communicate with other agents. The quality of ontologies are therefore crucial to the performance of intelligent agents. However, as with any other knowledge base, there is always the possibility that an ontology have some semantic defects. This paper presents an approach to ontology debugging using ideas borrowed from Truth Maintenance Systems (TMS).

2 Ontology Debugging

This section briefly introduces the key problems in ontology debugging and briefly outlines some of the major work in the field. Firstly, we provide basic definitions of incoherence and inconsistency of a DL-based ontology. An ontology is **incoherent** iff there is at least one unsatisfiable concept in its TBox. It is **inconsistent** iff there is no model for it. Generally speaking, the incoherence problem deals with concept-unsatisfiabilty within the TBox while an inconsistency problem also involves assertional axioms.[1] *Ontology debugging* is the process of identifying bugs and producing repair plans for an incoherent or inconsistent ontology. Most work has been done in ontology debugging is for the incoherence problem (i.e., debugging and repair of unsatisfiable concepts) although recently the problem of inconsistency has also been investigated.

Basically, the process of debugging ontology has two parts. The first is to identify which sets of axioms (or parts of axioms) are responsible for an unsatisfiable concept or an inconsistency. The next step is to propose how can these axioms be modified to

*The joint work with Natasha Alechina and Brian Logan. It has been published in [1].

[1] Note that the satisfiability checking problem in TBox can be reduced to a consistency checking problem by trying to construct a model for a concept using tableau rules.

make the concept satisfiable, or to restore consistency to the Knowledge Base (KB) with respect to some particular criteria.

Two main approaches to pinpointing problematic axioms have been proposed in the literature: glass-box and black-box. Glass-box methods, e.g., [2–5], use tableau-like rules to pinpoint the problematic axioms (concepts). These approaches obviously depend on a particular DL, as they have to modify the tableau rules to store and retrieve the sources of errors. Black-box methods on the other hand, e.g., [2, 6, 7], are reasoner-independent, since they only use the reasoner as an external component to diagnose whether an a concept is satisfiable with respect to a particular T-Box (or KB in the case of inconsistency problem).

3 Truth Maintenance Systems

Truth Maintenance Systems (TMS), e.g., [8], also known as Belief Revision Systems or Reason Maintenance Systems, play a central role in a style of belief revision called foundational belief revision. A TMS keeps track of dependencies between data to maintain the consistency of a database. A TMS consists of a set datum nodes and the justifications for them. A justification can be considered as a record of an inference, linking a datum node with the datum nodes used to derive it. Using these recorded dependencies, a TMS allows a problem-solver to quickly determine which nodes are "responsible" for belief in a particular datum.

According to [9], a TMS performs three main tasks: 1) given an assertion, find the assertions or assumptions used to derive it; 2) given a set of assumptions, find all the assertions can be derived from them; and 3) delete an assertion and all the consequences which have been derived from it. These tasks are also relevant to the problem of ontology debugging. For example, tracing the sources S_1 and S_2 of the assertions $A(x)$ and $\neg A(x)$, where A is a concept name and x is an individual in the ontology, gives the source of the contradiction (or clash) $S_1 \cup S_2$. Similarly, if one can find a minimal set of assumptions from which the contradictory assertions were derived, the minimal set of axioms which are the cause for the clash can also be identified.[2] This set corresponds to a *MUPS* in [5], or a *justification* for concept unsatisfiability defined in [10].

4 Using ATMS for Ontology Debugging

One particular type of TMS is an Assumption-based TMS (ATMS) [11]. In an ATMS, each node is associated with the set of sets of assumptions used to derive it, as well as the datum nodes that constitute its immediate antecedents. These sets of assumptions are termed *environments*, and are always kept minimal and consistent. In this way, backtracking is avoided and multiple solutions can be found at the same time.

In this section, we present an approach to ontology debugging using an ATMS. We focus on the problem of axiom pinpointing for an unsatisfiable concept (i.e., finding a set of axioms responsible for a unsatisfiably concept), and for simplicity, we only

[2] In the literature of ontology debugging, the idea of tagging an assertion with the axioms used to derive it has also been proposed in [3, 4].

consider the unfoldable $\mathcal{ALC}$ TBOX without disjunctions.[3] We show how the ATMS can be used to detect contradictions and to pinpoint sets of problematic axioms.

As a reasoner can easily detect that a concept is unsatisfiable by a satisfiability check, the key problem is to identify the sources of the unsatisfiability. This is the task of the ATMS. An ATMS node N_{datum} is of the form: $\langle datum, label, justifications \rangle$, where *datum* is an assertion such as $A_i(a)$, *label* is a set of environments (explained below), and *justifications* are the sets of nodes that directly derive N_{datum}. Since there are many ways a datum can be derived, it is possible to have multiple justifications for a particular node. The ATMS distinguishes two special types of datum nodes: assumptions and premises. *Assumptions* are foundational data. Each environment in the label of a (non-assumption) datum node comprises a set of assumptions from which the datum can consistently be derived. *Premises* are similar to assumptions, but are taken to hold universally, and are not explicitly represented in environments. The task of the ATMS is to ensure that each node label is consistent, sound, complete and minimal. As the reasoner informs the ATMS of new datum nodes and justifications, the ATMS *label propagation* algorithms update the labels of previously asserted nodes to remove any subsumed environments (in the case of a normal justification), or any environments which subsume an environment (in the case of a new justification for the distinguished node $N_\perp$ which represents contradiction).

The ontology debugging problem can be mapped onto the operations of the ATMS in a straightforward way. Each TBOX axiom is represented by an ATMS assumption. For concreteness, we assume a TBOX $\Gamma = \{ax_1, \ldots, ax_n\}$, where each axiom ax_i is of the form: $A_i \sqsubseteq C_i$ and all concept descriptions are in negation normal form (NNF). The assumption that each concept is non-empty, e.g., $A_i(c)$ for some constant c, is represented by an ATMS premise. The reasoner uses standard rules of inference to infer new concept instances from some consistent set of datum nodes (i.e., nodes whose labels do not subsume the label of $N_\perp$). A suitable list of rules that can be used by the reasoner to infer new justifications is shown in Figure 1. The process of creating the dependency graph terminates when no rule can be applied to any node of the graph. At this point, each environment of a node N_{datum} is a minimal set of axioms that can used to derive *datum*, and the label of $N_\perp$ consists of sets of axioms responsible for clashes.

5 Conclusion

In conclusion, there is a clear mapping between the functionality provided by the ATMS and the problems of ontology debugging, and we believe that a systematic investigation of the practicality of using an ATMS for ontology debugging is a fruitful direction for future research.

References

1. Nguyen, H.H., Alechina, N., Logan, B.: Ontology debugging with truth maintenance systems. In: Proceedings of ARCOE 2010. (2010)

[3]With disjunctions, the setting is more complicated. However, disjunctions can be handled in a number of different ways, e.g., similarly to the extended ATMS suggested by de Kleer in [12], or by using sub-graphs to deal with the or-branches.

$\sqsubseteq$-rule	If the current node is $N_{A_i(a)} :< A_i(a), L, J >$: if $N_{C_i(a)}$ exists, update its labels and justifications $L_{C_i(a)} = L_{C_i(a)} \cup L'$; $J_{C_i(a)} = J_{C_i(a)} \cup \{(A_i(a))\}$; otherwise, if $ax_i\ :\ A_i \sqsubseteq C_i \in TBOX$ then create a new node $N_{C_i(a)} :< C_i(a), L', \{(A_i(a))\} >$ where $L' = \{e \cup \{ax_i\} \| e \in L\}$
$\sqcap$-rule	If the current node is $N_{C_i(a) \sqcap C_j(a)} :< C_i(a) \sqcap C_j(a), L, J >$, then if $N_{C_i(a)}$ and $N_{C_j(a)}$ exist, update their labels and justifications; otherwise, create 2 new nodes $N_{C_i(a)} :< C_i(a), L, \{(C_i(a) \sqcap C_j(a))\} >$ $N_{C_j(a)} :< C_j(a), L, \{(C_i(a) \sqcap C_j(a))\} >$
$\exists$-rule	If the current node is $N_{(\exists s.C)(a)} :< (\exists s.C)(a), L, J >$, then create 2 nodes $N_{s(a,b)} :< s(a,b), L, \{(\exists s.C)(a)\} >$ $N_{C(b)} :< C(b), L, \{(\exists s.C)(a)\} >$ where b is a new individual.
$\forall$-rule	If the current node is $N_{(\forall s.C)(a)} :< (\forall s.C)(a), L, J >$ then if there exists $N_{s(a,b)}$, then create a node $N_{C(b)} :< C(b), L', \{((\forall s.C)(a), s(a,b))\} >$ where $L' = \{e_1 \cup e_2 \| e_1 \in L, e_2 \in L_{N_{s(a,b)}}\}$
$\bot$-rule	If there exist 2 nodes $N_{A_i(a)} :< A_i(a), L_1, J_1 >$ and $N_{\neg A_i(a)} :< \neg A_i(a), L_2, J_2 >$ in the graph, update $N_\bot$'s label and justifications as follows: $L_\bot = L_\bot \cup \{e_1 \cup e_2 \| e_1 \in L_1, e_2 \in L_2\}$, $J_\bot = J_\bot \cup \{j_1 \cup j_2 \| j_1 \in J_1, j_2 \in J_2\}$.

Fig. 1. Rules for creating and updating nodes in the dependency graph

2. Kalyanpur, A., Parsia, B., Sirin, E., Hendler, J.: Debugging unsatisfiable classes in OWL ontologies. Journal of Web Semantics **3**(4) (2005) 268–293
3. Lam, J.S.C., Sleeman, D.H., Pan, J.Z., Vasconcelos, W.W.: A fine-grained approach to resolving unsatisfiable ontologies. J. Data Semantics **10** (2008) 62–95
4. Meyer, T., Lee, K., Booth, R., Pan, J.Z.: Finding maximally satisfiable terminologies for the description logic ALC. In: Proceedings of AAAI 2006, AAAI Press (2006)
5. Schlobach, S., Cornet, R.: Non-standard reasoning services for the debugging of description logic terminologies. In: Proceedings of IJCAI'03, Morgan Kaufmann (2003) 355–360
6. Wang, H., Horridge, M., Rector, A., Drummond, N., Seidenberg, J.: Debugging OWL-DL ontologies: A heuristic approach. In: ISWC 2005, LNCS 3729, Springer (2005) 745–757
7. Horridge, M., Parsia, B., Sattler, U.: Explaining inconsistencies in OWL Ontologies. In: Proceedings of SUM '09, Berlin, Heidelberg, Springer-Verlag (2009) 124–137
8. Doyle, J.: A truth maintenance system. Artificial Intelligence **12**(3) (1979) 231–272
9. Shapiro, S.C.: Belief revision and truth maintenance systems: An overview and a proposal. Technical report, SUNY-Buffalo (1998)
10. Kalyanpur, A., Parsia, B., Grau, B., Sirin, E.: Justifications for entailments in expressive description logics. Technical report, University of Maryland (2006)
11. de Kleer, J.: An assumption-based tms. Artificial Intelligence **28**(2) (March 1986) 127–162
12. de Kleer, J.: Extending the ATMS. Artificial Intelligence **28**(2) (1986) 163–196