On direct debugging of aligned ontologies*

Kostyantyn Shchekotykhin, Philipp Fleiss, Patrick Rodler, and Gerhard Friedrich

Alpen-Adria Universität, Klagenfurt, 9020 Austria
firstname.lastname@aau.at

Abstract. Modern ontology debugging methods allow efficient identification and localization of faulty axioms defined by a user while developing an ontology. However, in many use cases such as ontology alignment the ontologies might include many conflict sets, i.e. sets of axioms preserving the faults, thus making the ontology diagnosis infeasible. In this paper we present a debugging approach based on a direct computation of diagnoses that omits calculation of conflict sets. The proposed algorithm is able to identify diagnoses for an ontology which includes a large number of faults and for which application of standard diagnosis methods fails. The evaluation results show that the approach is practicable and is able to identify a fault in adequate time.

1 Motivation and algorithm details

Ontology development and maintenance relies on an ability of users to express their knowledge in form of logical axioms. However, the knowledge acquisition process might be problematic since a user can make a mistake in an axiom being modified or a correctly specified axiom can trigger a hidden bug in an ontology. These bugs might be of a different nature and result in violation of such *requirements* as consistency of an ontology or satisfiability of its classes. In such scenarios as ontology matching the complexity of faults might be very high because multiple disagreements between ontological definitions and/or modeling problems can be triggered by aliments at once.

Ontology debuggers simplify the development by allowing their users specification of requirements to the intended (target) ontology. In addition, a user can provide $\mathcal{B}$ set of *background* (correct) axioms and sets of positive P and negative N test cases. A *positive* test case is a set of axioms that must be entailed by the intended ontology, whereas a *negative* test case must not. If any of the requirements or test cases are broken, i.e. an ontology $\mathcal{O}$ is *faulty*, then the tuple $\langle \mathcal{O}, \mathcal{B}, P, N \rangle$ is a *diagnosis problem instance*. For a given problem instance a debugging tool computes a set of axioms $\mathcal{D} \subseteq \mathcal{O}$ called *diagnosis*. An expert should remove or modify at least *all* axioms of a diagnosis in order to be able to formulate the *target ontology* $\mathcal{O}_t$.

Nevertheless, in real-world scenarios debugging tools can return a set of alternative diagnoses $\mathbf{D}$. The reason is the practical impossibility for a user to specify such a set of requirements and test cases, prior to a debugging session, providing all information required for identification of the *target diagnosis* $\mathcal{D}_t$, i.e. the diagnosis which application allows formulation of the intended ontology. *Diagnosis discrimination* methods allow their users to reduce the number of diagnoses to be considered. An interactive algorithm suggested in [4] identifies the target diagnosis by asking an oracle a sequence of questions: whether some axiom is entailed by the target ontology or not. A general

* This research is funded by Austrian Science Fund (Project V-Know, contract 19996).

interactive ontology diagnosis algorithm can be described as follows: (1) Generate a set of diagnoses $\mathbf{D}$ including at most n diagnoses. (2) Compute a set of queries and select the best one using a predefined measure. (3) Ask the oracle and, depending on the answer, add the query either to P or to N. (4) Update the set of diagnoses $\mathbf{D}$ and remove the ones that do not comply with the newly acquired test case. (5) Update the tree and repeat from Step 1 if the tree contains open nodes. (6) Return the set of diagnoses $\mathbf{D}$. The resulting set $\mathbf{D}$ includes only diagnoses that are not differentiable in terms of their entailments, but have some syntactical differences. The preferred target diagnosis $\mathcal{D}_t$, in this case, should be selected by the user using some ontology editor such as Protégé.

Most of the debugging approaches, including [4], follow the standard model-based diagnosis approach [3] and compute diagnoses using *conflict sets* CS, i.e. irreducible sets of axioms $ax_i \in \mathcal{O}$ that preserve violation of at least one requirement. The computation of one conflict set can be done within a polynomial number of calls to the reasoner, e.g. by QUICKXPLAIN algorithm [2]. To identify a diagnosis of cardinality $|\mathcal{D}| = m$ the hitting set algorithm suggested in [3] requires computation of m conflict sets. In some practical scenarios, such ontology matching, the number of conflict sets m can be large, thus making the ontology debugging practically infeasible.

In this paper we present two algorithms INV-HS-TREE and INV-QUICKXPLAIN, which inverse the standard model-based approach to ontology debugging and compute diagnoses directly, rather than by means of conflict sets (see [5] for a detailed description of the algorithms). The main function of the latter algorithm splits recursively the initial diagnosis problem instance into two sub-problems by partitioning the axioms of a given faulty ontology into two subsets. In many cases SPLIT simply partitions the set of axioms into two sets of equal cardinality. The algorithm continues to divide diagnosis problems until it identifies a set $\mathcal{D}'$ such that $\mathcal{O} \setminus \mathcal{D}'$ fulfills all the requirements, but $\mathcal{O} \setminus \mathcal{D}'_i$, where $\mathcal{D}'_i$ are partitions of $\mathcal{D}'$, not. In further iterations the algorithm minimizes the $\mathcal{D}'$ by splitting it into sub-problems of the form $\mathcal{D}' = \mathcal{D} \cup \mathcal{O}_\Delta$, where $\mathcal{O}_\Delta$ contains only one axiom. In the case when $\mathcal{D}'$ is a diagnosis and $\mathcal{D}$ is not, the algorithm decides that $\mathcal{O}_\Delta$ is a subset of the sought diagnosis. Just as the original algorithm, INV-QUICKXPLAIN always terminates and returns either a diagnosis $\mathcal{D}$ or "no diagnosis".

In order to enumerate all possible diagnoses we modified the HS-TREE algorithm [3] to accept diagnoses as node labels instead of conflict sets. The INV-HS-TREE algorithm constructs a directed tree from root to the leaves, where each node nd is labeled either with a diagnosis $\mathcal{D}(nd)$ or ✓ (closed) or × (pruned). The latter two labels indicate that the node cannot be extended. Each edge outgoing from the open node nd is labeled with an element $s \in \mathcal{D}(nd)$. $HS(nd)$ is a set of edge labels on the path from the root to the node nd. Initially the algorithm creates an empty root node and adds it to the *queue*, thus, implementing a breadth-first search strategy. Until the queue is empty, the algorithm retrieves the first node nd from the queue and labels it with either:

1. × if there is a node nd', labeled with either ✓ or ×, such that $H(nd') \subseteq H(nd)$ (pruning non-minimal paths), or
2. $\mathcal{D}(nd')$ if a node nd' exists such that its label $\mathcal{D}(nd') \cap H(nd) = \emptyset$ (reuse), or
3. $\mathcal{D}$ if $\mathcal{D}$ is a diagnosis for the diagnosis problem instance $\langle \mathcal{O}, \mathcal{B} \cup H(nd), P, N \rangle$ computed by INV-QUICKXPLAIN (compute), or
4. ✓ (closed).

The leaf nodes of a complete tree are either pruned (×) or closed (✓) nodes.

In the diagnosis discrimination settings the ontology debugger acquires new knowledge that can invalidate some of the diagnoses labeling the tree nodes. During the tree

update INV-HS-TREE searches for the nodes with invalid labels, removes its label and places it to the list of open nodes. Moreover, the algorithm removes all the nodes of a subtree originating from this node. After all nodes with invalid labels are cleaned-up, the algorithm attempts to reconstruct the tree by reusing the remaining valid minimal diagnoses (rule 2, INV-HS-TREE). Such aggressive pruning of the tree is feasible since a) the tree never contains more than n nodes that were computed with INV-QUICKXPLAIN and b) computation of a possible modification to the minimal diagnosis, that can restore its validness, requires invocation of INV-QUICKXPLAIN and, therefore, as hard as computation of a new diagnosis. Note also, that in a common diagnosis discrimination setting n is often set to a small number, e.g. 10, in order to achieve good responsiveness of the system. In the direct approach this setting limits the number of calls to INV-QUICKXPLAIN to n and results in a small size of the search tree. The latter is another advantage of the direct method as it requires much less memory in comparison to a debugger based on original HS-TREE.

2 Evaluation

We evaluated the direct ontology debugging technique using aligned ontologies generated in the framework of OAEI 2011 [1]. These ontologies represent a real-world scenario in which a user generated ontology alignments by means of (semi-)automatic tools. All ontologies used in the experiments are available online, together with detailed evaluation results[1].

In the first experiment we applied the debugging technique to the set of aligned ontologies resulting from "Conference" set of problems, which is characterized by lower precision and recall of the applied systems (the best F-measure 0.65) in comparison, for instance, to the "Anatomy" problem (average F-measure 0.8). The Conference test suite includes 286 ontology alignments generated by 14 ontology matching systems including: a) 140 ontologies are consistent and coherent; b) 122 ontologies are incoherent; c) 26 ontologies are inconsistent; and in 8 cases a reasoner was unable to classify in two hours[2]. Note that only two systems `CODI` and `MaasMtch` out of 14 were able to generate consistent and coherent alignments. This observation confirms the importance of high-performance ontology debugging methods.

The 146 ontologies of the cases b) and c) were analyzed with both HS-TREE and INV-HS-TREE. The results of the experiment show that for 133 ontologies out of 146 both approaches were able to compute the required amount of diagnoses. HT-TREE required 49, 61 and 80 sec. on average to find 1, 9 and 30 leading minimal diagnoses. The direct algorithm required less time and returned the requested number of diagnoses on average in 27, 52 and 72 sec. respectively. In the 13 cases HS-TREE was unable to find all requested diagnoses in each experiment. Within 2 hours the algorithm calculated only 1 diagnosis for `csa-conference-ekaw` and for `ldoa-conference-confof` it was able to find 1 and 9 diagnoses, whereas INV-HS-TREE required 9 sec. for 1, 40 sec. for 9 and 107 sec. for 30 diagnoses on average.

Moreover, in the first experiment we evaluated the efficiency of the interactive direct debugging approach applied to the cases listed in Table 1. We selected the target diagnosis randomly among all diagnoses of the following diagnosis problem instance $\langle M_f, \mathcal{O}_i \cup \mathcal{O}_j \cup M_t, \emptyset, \emptyset \rangle$, where M_f and M_t are the sets of *false* and *true* positive

[1] http://code.google.com/p/rmbd/wiki/DirectDiagnosis

[2] Core-i7 (3930K) 3.2Ghz, 32GB RAM running Ubuntu 11.04, Java 6 and HetmiT 1.3.6.

Matcher	Ontology 1	Ontology 2	Time (sec)	#Query	React (sec)	#CC	CC (ms)
ldoa	conference	confof	11.6	6	1.4	430	3
ldoa	cmt	ekaw	48.5	21	2.2	603	16
mappso	confof	ekaw	9.9	5	1.8	341	7
optima	conference	ekaw	16.7	5	2.5	553	8
optima	confof	ekaw	23.9	20	1.1	313	14
ldoa	conference	ekaw	56.6	35	1.4	253	53
csa	conference	ekaw	6.7	2	2.7	499	3
mappso	conference	ekaw	27.4	13	1.8	274	28
ldoa	cmt	edas	24.7	22	1.1	303	8
csa	conference	edas	18.4	6	2.7	419	5
csa	edas	iasted	1744.6	3	349.2	1021	1333
ldoa	ekaw	iasted	23871.4	10	1885.9	287	72607
mappso	edas	iasted	18400.2	5	2028.2	723	17844

Table 1. Diagnosis discrimination using direct ontology debugging and Entropy scoring function. **React** time required to compute 9 diagnoses and a query, **#CC** number of consistency checks, **CC** gives average time needed for one consistency check.

alignments computed from the set of correct alignments M_c, provided by the organizers of OAEI 2011, and the set M_{ij} generated by a ontology matching system. In the experiment the used the Entropy scoring function [4] with prior fault probabilities of axioms corresponding to aliments set to $1 - v$, where v is the confidence value of the matcher for an axiom. All axioms of $\mathcal{O}_i$ and $\mathcal{O}_j$ were assumed to be correct and were assigned small probabilities. The results presented in Table 1 were computed for the diagnosis problem instance $\langle M_{ij}, \mathcal{O}_i \cup \mathcal{O}_j, \emptyset, \emptyset \rangle$. The experiment shows that the system was able to identify the target diagnosis efficiently with small reaction times. The system's performance decreased only in the cases when a reasoner required much time to verify the consistency of an ontology.

In the second scenario we applied the direct method to unsatisfiable and classifiable within 2 hours ontologies, generated for the Anatomy problem. The source ontologies $\mathcal{O}_1$ and $\mathcal{O}_2$ include 11545 and 4838 axioms correspondingly, whereas the size of the alignments varies between 1147 and 1461 axioms. The target diagnosis selection process was performed in the same way as in the first experiment. The results of the experiment show that the target diagnosis can be computed within 40 second in an average case. Moreover, INV-HS-TREE slightly outperformed HS-TREE.

References

1. Euzenat, J., Ferrara, A., van Hage, W.R., Hollink, L., Meilicke, C., Nikolov, A., Ritze, D., Scharffe, F., Shvaiko, P., Stuckenschmidt, H., Sváb-Zamazal, O., dos Santos, C.T.: Final results of the Ontology Alignment Evaluation Initiative 2011. In: Proceedings of the 6th International Workshop on Ontology Matching. pp. 1–29. CEUR-WS.org (2011)
2. Junker, U.: QUICKXPLAIN: Preferred Explanations and Relaxations for Over-Constrained Problems. In: Proc. 19th National Conference on Artificial Intelligence. pp. 167–172 (2004)
3. Reiter, R.: A Theory of Diagnosis from First Principles. Artificial Intelligence 32(1), 57–95 (1987)
4. Shchekotykhin, K., Friedrich, G., Fleiss, P., Rodler, P.: Interactive ontology debugging : two query strategies for efficient fault localization. Web Semantics: Science, Services and Agents on the World Wide Web 12-13, 88–103 (2012)
5. Shchekotykhin, K., Friedrich, G., Fleiss, P., Rodler, P.: Direct computation of diagnoses for ontology debugging. arXiv 1–16 (2012) http://arxiv.org/abs/1209.0997