On computing minimal conflicts for ontology debugging

Kostyantyn Shchekotykhin and **Gerhard Friedrich**[1] and **Dietmar Jannach**[2]

Abstract. Ontology debugging is an important stage of the ontology life-cycle and supports a knowledge engineer during the ontology development and maintenance processes. Model based diagnosis is the basis of many recently suggested ontology debugging methods. The main difference between the proposed approaches is the method of computing required *conflict sets*, i.e. a sets of axioms such that at least one axiom of each set should be changed (removed) to make ontology coherent. Conflict set computation is, however, the most time consuming part of the debugging process. Consequently, the choice of an efficient conflict set computation method is crucial for ensuring the practical applicability of an ontology debugging approach.

In this paper we evaluate and compare two popular minimal conflict computation methods: QUICKXPLAIN and SINGLE_JUST. First, we analyze best and worst cases of the required number of coherency checks of both methods on a theoretical basis assuming a black-box reasoner. Then, we empirically evaluate the run-time efficiency of the algorithms both in black-box and in glass-box settings.

Although both algorithms were designed to view the reasoner as a black box, the exploitation of specific knowledge about the reasoning process (glass-box) can significantly speed up the run-time performance in practical applications. Therefore, we present modifications of the original algorithms that can also exploit specific data from the reasoning process.

Both a theoretical analysis of best- and worst-case complexity as well as an empirical evaluation of run-time performance show that QUICKXPLAIN is preferable over SINGLE_JUST.

1 MOTIVATION

With an increasing number of applications that rely on ontologies, these knowledge bases are getting larger and more complex. Thus, corresponding knowledge bases can include definitions of thousands of concepts and roles from different domains. RDF search engines like Watson [2] for instance facilitate the creation of composite ontologies that reuse the definition of concepts and roles published on the Web. Moreover, the community of ontology users is getting more heterogeneous and nowadays includes many members from various industrial and scientific fields. Hence, different faults can be easily introduced during creation and maintenance of ontologies.

Recent debugging methods as described in [4, 8, 9, 11] help the user to localize, understand, and correct faults in ontologies and are already implemented in popular ontology development tools like Protégé[3] or Swoop[4].

All currently suggested approaches for ontology debugging aim at the automated computation of a set of changes to the ontology that restore the coherence of its terminology (*diagnosis*). In order to accomplish this task efficiently, current diagnosis approaches are based on the computation of axiom subsets that define an incoherent terminology (conflict sets).

Diagnosis techniques: Currently, two approaches are used for the computation of diagnoses in ontology debugging: *Pinpointing* [12] and *Reiter's model-based diagnosis* (MBD) [10]. *Pinpoints* are used to avoid the computation of minimal hitting sets of conflict sets by approximating minimal diagnoses by their supersets. However, the pinpoints themselves are computed on the basis of *all* minimal conflicts. In contrast, in MBD approaches (minimal) conflicts are computed *on demand* and diagnoses are computed with increasing cardinality by constructing a hitting-set tree (HSTREE). Consequently, this method will find those diagnoses first that suggest minimal changes and avoids both the computation of very implausible multi-fault diagnoses and the costly computation of the set of *all* minimal conflicts. Note that Reiter's original proposal does not work correctly for non-minimal conflicts [5] and shows limited performance for non-minimal conflicts. A modified diagnosis method was however introduced in [4] which avoids these deficits. The general question whether pinpoints or leading diagnoses are more appropriate as an output of the debugging process is still open.

Conflict computation: Current approaches like SINGLE_JUST[8] and QUICKXPLAIN[4] either treat the underlying reasoner as a black-box or a glass-box.

In (invasive) glass-box approaches the developer of the debugging method can exploit specifics of the theorem prover. In [9], for instance, a conflict computation approach was proposed which requires modifications of existing reasoning algorithms as its aim is to compute sets of conflicts during the reasoning process as efficiently as possible. The main drawback of such glass-box approaches however is that they can be used only for a particular description logic [1], like $\mathcal{SHOIN}(\mathcal{D})$ [9]. Hence, only a particular reasoner (or even a version of a reasoner) and a particular type of logic can be used. Moreover, glass-box modifications to reasoning systems often remove existing optimizations and thus are typically slower then their non-modified analogues. In addition, glass-box approaches to conflict set minimization do not guarantee the minimality of returned conflict sets and further (black-box) minimization is required [8].

On the other hand, black-box algorithms are completely independent from the reasoning process and just use the boolean outputs of the theorem prover. These algorithms are therefore logic-independent and can exploit the full power of highly optimized reasoning methods. Still, in case of an unsatisfiable set of axioms, all the axioms are considered as a conflict since no further information is available. Conflicts are typically minimized by additional calls to a theorem prover. In order to make black-box approaches applicable in cases where theorem proving is expensive, the number of such calls must

[1] University Klagenfurt, Austria, email: firstname.lastname@ifit.uni-klu.ac.at
[2] Dortmund University of Technology, Germany, email: dietmar.jannach@udo.edu
[3] http://www.co-ode.org
[4] http://code.google.com/p/swoop/

be minimized.

In current systems, two main approaches for conflict set computation are used, SINGLE_JUST [8] and QUICKXPLAIN [4]. In general, both of them can be used in glass-box and black-box settings. In this paper we show that QUICKXPLAIN is preferable over SINGLE_JUST in both settings based on a theoretical analysis of best and worst cases and an empirical performance evaluation for a simulated average case. In addition, we propose modifications to the original algorithms to further improve the run-time performance in glass-box settings.

The reminder of the paper is organized as follows, Section 2 provides theoretical study of conflict set computation methods and includes a brief description of the main algorithms as well as the analysis of extreme cases. In Section 3 we present the results of empirical evaluation of QUICKXPLAIN and SINGLE_JUST in both black- and glass-box settings. The paper closes with a discussion of the future work.

2 COMPUTING MINIMAL CONFLICTS

We will focus on the comparison of two popular algorithms QUICKXPLAIN [6] and SINGLE_JUST [8]. The presented comparison is possible because application scenarios and strategies of these algorithms are similar. Both methods are designed to compute only one minimal conflict set per execution. The combinations of QUICKXPLAIN + HSTREE [4] and SINGLE_JUST + HSTREE (also referred as ALL_JUST) [8] are used to obtain a set of minimal diagnoses diagnoses or to enumerate minimal conflict sets (justifications). Therefore, QUICKXPLAIN and SINGLE_JUST can be compared both theoretically and empirically.

Algorithm: QUICKXPLAIN(B, C)
Input: trusted knowledge B, set of axioms C
Output: minimal conflict set CS

---

(1) **if** $isCoherent(B \cup C)$ or $C = \emptyset$ **return** $\emptyset$;
(2) $AX \leftarrow getFaultyAxioms(C)$;
(3) **if** $AX \neq \emptyset$ **then** $C \leftarrow C \cap AX$;
(4) **return** $computeConflict(B, B, C)$

function $computeConflict(B, \Delta, C)$
(5) **if** $\Delta \neq \emptyset$ and not $isCoherent(B)$ **then return** $\emptyset$;
(6) **if** $|C| = 1$ **then return** C;
(7) int $n \leftarrow |C|$; int $k := split(n)$
(8) $C_1 \leftarrow \{ax_1, \ldots, ax_k\}$ and $C_2 := \{ax_{k+1}, \ldots, ax_n\}$;
(9) $CS_1 \leftarrow computeConflict(B \cup C_1, C_1, C_2)$;
(10) **if** $CS_1 = \emptyset$ **then** $C_1 \leftarrow getFaultyAxioms(C_1)$;
(11) $CS_2 \leftarrow computeConflict(B \cup CS_1, CS_1, C_1)$;
(12) **return** $CS \leftarrow CS_1 \cup CS_2$;

---

function $getFaultyAxioms(C)$
(13) $AX \leftarrow getConflictSet\_glassBox()$;
(14) **if** $AX = \emptyset$ **then return** C;
(15) **else return** AX;

Figure 1. Generalized QUICKXPLAIN algorithm

QUICKXPLAIN. This algorithm (listed in Figure 1) takes two parameters as an input, B a set of axioms that are considered as correct by a knowledge engineer and C a set of axioms, which should be analyzed by the algorithm. QUICKXPLAIN follows a *divide-and-conquer* strategy and splits the input set of axioms C into two subsets C_1 and C_2 on each recursive call. If the conflict set is a subset of either C_1 or C_2, the algorithm significantly reduces the search space. If for instance the *splitting function* is defined as $split(n) = n/2$ then the search space will be reduced in half just with one call to a reasoner. Otherwise, the algorithm re-adds some axioms $ax \in C_2$ to C_1. With the splitting function defined above, the algorithm will add a half of all axioms of the set C_2.

The choice of the splitting function is crucial since it affects the number of required coherency checks. The knowledge engineer can define a very effective splitting function for a concrete problem, e.g., if there exists some a priori knowledge about faulty axioms of an ontology. However, in the general case it is recommended to use the function that splits the set C of all axioms into two subsets of the same size since the path length from the root of the recursion tree to a leaf will contain at most $\log_2 n$ nodes. Thus, if the cardinality of the searched minimal conflict set $|CS| = k$ in the best case, i.e. when all k elements belong to a single subset C_1, the number of required coherency checks is $\log_2 \frac{n}{k} + 2k$. The worst case for QUICKXPLAIN is observed when the axioms of a minimal conflict set always belong to different sets C_1 and C_2, i.e., if for instance a minimal conflict set has two axioms and one is positioned at the first place of set C and the other one at the last. In this case the number of coherency checks is $2k(\log_2 \frac{n}{k} + 1)$ [6].

Note that we modified the original QUICKXPLAIN algorithm (Figure 1) such that it can be used with both black- and glass-box approaches. The algorithm issues two types of calls to a reasoner, $isCoherent(T)$ and $getConflictSet\_glassBox()$. The first function returns *true* if the given terminology T is coherent. $getConflictSet\_glassBox()$ returns a set of axioms AX that are responsible for incoherence ($CS \subseteq AX$). This function can only be used if the reasoner supports glass-box debugging. If this is the case the reasoner will able to return the set AX which was generated during the reasoning process. If only black-box usage is possible then a practical implementation of an ontology debugger should override this function with one that returns an empty set. In this case the modified algorithm is equal to the original one given in [6].

Moreover, the first part of the algorithm (lines 1-4) is required to check if an ontology is actually incoherent. This check is required for two reasons. First, the result of conflict set computation for an already coherent ontology using a reasoner as a black-box will include all axioms of this ontology; second, a feedback of a glass-box reasoner executed at this stage can significantly reduce the search space of QUICKXPLAIN. The same can also be noted for SINGLE_JUST.

SINGLE_JUST This algorithm (see Figure 2) follows an *expand-and-shrink* strategy and has two main loops. The first one creates a set of CS that includes all axioms of the minimal conflict set and the second one minimizes CS by removing axioms that do not belong to the minimal conflict set.

The algorithm includes two functions $select(T)$ and $fastPruning(T)$ that can be tuned to improve its performance. The first function starts by selecting a predefined number of axioms num from the given set. The number of axioms that are selected can grow with a certain factor f (see [7]). The $fastPruning$ function implements a pruning strategy for CS with a sliding window technique. The pruning algorithm takes the size of the window

Algorithm: SINGLE_JUST(B, C)
Input: trusted knowledge B, set of axioms C
Output: minimal conflict set CS

(1) **if** $isCoherent(B \cup C)$ or $C = \emptyset$ **return** $\emptyset$;
(2) $AX \leftarrow getFaultyAxioms(C)$;
(3) **if** $AX \neq \emptyset$ **then** $C \leftarrow C \cap AX$;
(4) **return** $computeConflict(B, C)$

function $computeConflict(B, C)$
(5) $CS \leftarrow B$;
(6) **do**
(7) $CS \leftarrow CS \cup select(C \setminus CS)$;
(8) **while** ($isCoherent(CS)$);
(9) $CS \leftarrow fastPruning(getFaultyAxioms(CS))$;
(10) **for each** $ax \in CS$ **do**
(11) $CS \leftarrow CS \setminus \{ax\}$
(12) **if** $isCoherent(CS)$ **then** $CS \leftarrow CS \cup \{ax\}$;
(13) **else** $CS \leftarrow getFaultyAxioms(CS)$;
(14) **return** CS;

Figure 2. Generalized SINGLE_JUST algorithm

$window$ and the set of axioms CS as an input and outputs a set of axioms $CS' \subseteq CS$. In the form it was implemented in OWL-API[5], the pruning algorithm partitions the input set CS with n axioms into $p = n/window$ parts $P_i, i = 1, \ldots, p$ and then sequentially tests coherency of each set $CS_i = CS \setminus P_i, i = 1, \ldots, p$. Note also that OWL-API includes two variants of the pruning method, one with constant and one with shrinking window size. In further analysis and evaluation we will consider only the variant with the constant window size.

Let us consider the best and worst cases for SINGLE_JUST. In the best case, all axioms of a minimal conflict set CS belong to some partition set P_i. Thus, given an axioms set C of cardinality n that contains a minimal conflict set CS of cardinality k, the algorithm will make at most $1 + p + min(window, num)$ coherency checks. In the worst case, the first iteration will require at least $log_f\left(1 - \frac{1-f}{num}n\right)$ coherency checks and both the sliding window and final minimization $p + min(k/p, 1)n$ checks, if all k axioms of the minimal conflict set belong to different partitions P_i.

The theoretical analysis of the two algorithms thus shows that QUICKXPLAIN has a smaller interval of possible number of coherency checks in comparison to SINGLE_JUST (see Figure 3)[6].

Note also that the interaction with the reasoner used in SINGLE_JUST in Figure 2 is organized in the same way as in QUICKXPLAIN, i.e., by means of the functions $isConsistent$ and $getFaultyAxioms$. However, if a black-box approach to ontology debugging is used, the modified algorithm presented on the 2 is equal in terms of number of consistency checks to the original one suggested in [8]. Moreover, both generalized algorithms can also be used to detect conflict sets that cause unsatisfiability of a certain concept. This is possible if we introduce one more input parameter $Concept$ and rewrite the coherency checking function such that $isCoherent(C, Concept)$ returns $false$ if $Concept$ is unsatisfiable with respect to the terminology defined by a set of axioms C. Otherwise this function should return $true$.

[5] Unfortunately, the authors of the SINGLE_JUST algorithm did not provide a specification of the fast pruning method neither in [7] nor in [8]. Therefore we analyzed the OWL-API (http://owlapi.sourceforge.net/ checked on June 7, 2008) implementation that is referred by the authors in [8].

[6] The values of SINGLE_JUST parameters were taken from the OWL-API implementation.

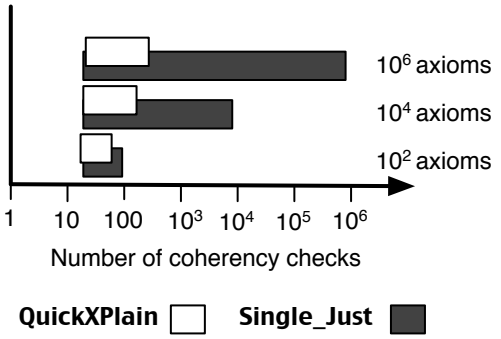

Figure 3. Intervals for numbers of possible coherency checks required to identify a minimal conflict set of cardinality $k = 8$ in an ontology of n axioms. QUICKXPLAIN parameters: $split(n) = n/2$. SINGLE_JUST: number of axioms on the first iteration $num = 50$, increment factor $f = 1.25$, window size $window = 10$.

The algorithms presented on Figures 1 and 2 can also exploit the structural relations between axioms by means of specifically implemented functions $split$, $select$ and $fastPruning$. One can thus for instance select and/or partition axioms so that axioms with intersecting sets of concepts will be considered first.

3 EMPIRICAL EVALUATION

The theoretical analysis of the algorithms showed that QUICKXPLAIN is preferable over SINGLE_JUST since it has much lower variation of the number of required reasoner calls. Nevertheless, the extremum conditions of the discussed best and worth cases are rather specific. Therefore, an analysis of the average case has to be done in order to make the comparison complete. However, evaluating this case is problematic, since there are no publicly available collections of incoherent ontologies that are published on the Web and are suitable for such tests. Moreover, there is no a priori knowledge on the distribution of conflicts. In other words, we do not know how the faulty axioms are most often positioned in an ontology. Therefore, we simulated the occurence of faults in the ontology in order to obtain a measure of the numbers of coherency checks required by QUICKXPLAIN and SINGLE_JUST. These statistics can be then used to calculate the average number of required coherency checks. Moreover, for our purposes it is enough to generate and then compute only one conflict set, since none of the analyzed algorithms can improve its performance on subsequent executions by using data from the previous runs.

The test case generation method was designed under the following assumptions: (1) All axioms have the same probability to be part of a conflict set (uniform distribution). Thus, for an ontology with n axioms, the probability for each axiom to be a source of a conflict is $1/n$. (2) The cardinalities of minimal conflict sets follow the binomial distribution with the number of trials t equal to the maximal length of the dependency path from an axiom selected according to the first assumption to all axioms that directly or indirectly depend on concepts declared in the selected axiom. The value t corresponds to the maximum possible cardinality of a conflict that can be generated for a selected axiom. The success probability, which is the second

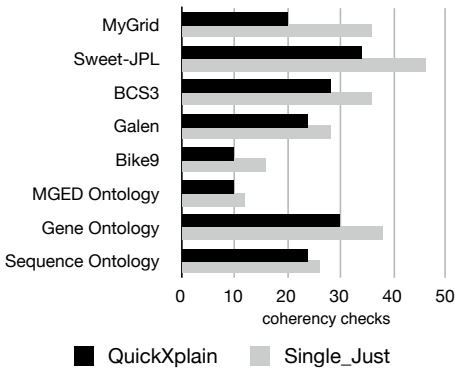

Figure 4. Average number of consistency checks for QUICKXPLAIN and SINGLE_JUST using reasoner as a black-box

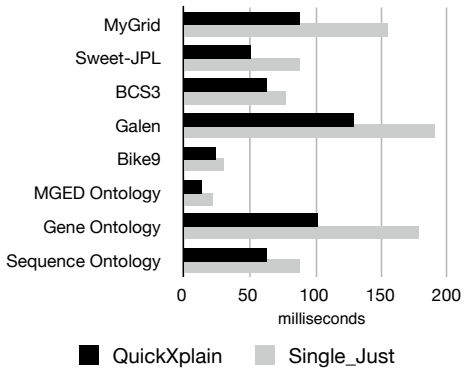

Figure 5. Average running times for black-box QUICKXPLAIN and SINGLE_JUST

parameter of the distribution, is set to $1/t$. Hence minimal conflict sets of smaller cardinality are more likely to appear.

The process starts with the generation of a uniformly distributed number i for the interval $[1, n]$. This number corresponds to an axiom ax_i that is the initial axiom of a conflict set. Then the algorithm queries a reasoner for a taxonomy of a given ontology to find out all axioms that contain concept definitions that are either directly or indirectly dependent on a concept defined in ax_i. The length of a longest dependency path t is then used to generate the value c, which corresponds to the minimal cardinality of a conflict set according to the second assumption. Next, we retrieve all axioms which define concepts that subsume one of the concepts declared in ax_i such that subsumption is made over the $c - 1$ other concepts $(C_1 \sqsubseteq ... \sqsubseteq C_{c-1} \sqsubseteq C_c)$. If more then one axiom is retrieved then we select randomly one of them (denoted as ax_j). Both axioms are modified to create a conflict (e.g. by inserting a new concept definition in ax_i and its negation in ax_j). Thus, the generation method generates faults that correspond to local or propagated causes of unsatisfiability that were observed by Wang et al [14] in many real-world ontologies.

Note that the real cardinality of a minimal conflict is unknown prior to the execution of a conflict compution algorithm, since we do not investigate all possible dependencies of the modified axioms.

In the tests we used Pellet 1.5.1[7] as a reasoner and the SSJ statistical library[8] to generate the required random numbers. As can be seen in Figure 4 and Figure 5, in the average case (after 100 simulations) QUICKXPLAIN outperformed SINGLE_JUST in all eight test ontologies MyGrid (8179 axioms), Sweet-JPL (3833 axioms), BCS3 (432 axioms), Galen (3963 axioms), MGED Ontology (236), Bike9 (215), Gene Ontology (1759) and Sequence Ontology (1745). In this test we measured both the number of checks and the elapsed time.

Note also that the results that we obtained when using Pellet can in generally also be transferred to other reasoners that – in these settings have shown to have comparable performance [13]. All experiments have been performed on a MacBookPro (Intel Core Duo) 2 GHz with 2 GB RAM and 1.5 GB maximum Java memory heap size.

Beside using the reasoner as a black-box, both variants of QUICKXPLAIN and SINGLE_JUST can also be used in glass-box settings. However, the theoretical analysis of these cases is not trivial, since we cannot predict the number of axioms that will be returned by a glass-box method on each iteration of the conflict computation algorithm. The computed conflict set can include extra axioms that do not belong to the searched minimal conflict set because of non-determinisms of a reasoning algorithm such as max-cardinality restrictions or special features of the tracing algorithm itself. Therefore, our empirical evaluation of different combinations of QUICKXPLAIN and SINGLE_JUST is based on two different glass-box implementations: invasive [9] and naïve non-invasive.

In general, all glass-box methods implement tracing of axioms that were used by the reasoner to prove the unsatisfiability of a concept. The invasive method first stores all correspondences between source axioms and internal data structures of the reasoner and tracks the changes in internal data structures during the normalization and absorption phases (see [9] for details). In [9], it is also suggested to add tracing to the $\mathcal{SHOIN}(\mathcal{D})$ tableaux expansion rules to enable a very precise tracking of axioms so that the resulting axioms set will be as small as possible. The main drawback of this approach however is that such a modification disables many key optimizations, which are critical for the excellent performance of modern OWL reasoners [8].

In the non-invasive approach that we developed for our evaluation, we only track which concepts were unfolded by the reasoner and then search for all axioms in which these concepts are defined using the OWL-API. This method does not analyze the details of the reasoning process and thus, the resulting set of axioms is only in the best case equals to the set returned by the invasive method. However, such an approach can have a shorter execution time, since it does not require changes in the optimized reasoning process except for the insertion of a logging method for unfolded concepts.

Pellet 1.5.1 already includes an implementation of the invasive method (explanations of clashes) and can also be configured to turn on logging which is required for the non-invasive method. The only modification to the reasoner was to add a fast fail behavior in the satisfiability check. By default, Pellet searches for all unsatisfiable concepts. However, for the minimal conflict set computation algorithm it is enough to find just one such concept, since in this case the terminology is already incoherent.

We performed the tests of the glass-box methods using the same test bundle that was used for the black-box tests. The evaluation shows that QUICKXPLAIN is faster in both approaches (see Figures 6 and 7). When using the feedback from the glass-box satisfiability check, QUICKXPLAIN performed better then SINGLE_JUST in all

[7] http://pellet.owldl.com/
[8] http://www.iro.umontreal.ca/∼simardr/ssj/indexe.html

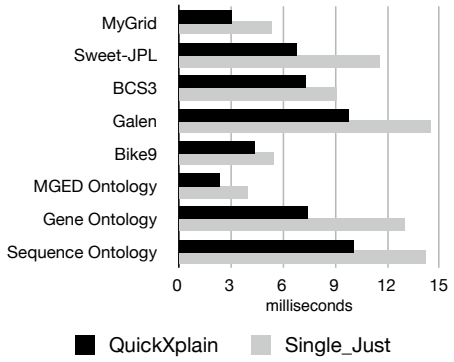

Figure 6. Average running times for non-invasive glass-box

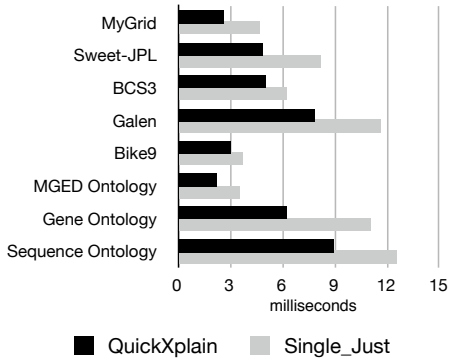

Figure 7. Average running times for invasive glass-box

the cases. Note also that the difference in the average running times for QUICKXPLAIN and SINGLE_JUST in invasive and non-invasive glass-box settings is not significant as both glass-box methods can in general reduce the search space of minimal conflict computation algorithms very rapidly.

4 CONCLUSIONS & FUTURE WORK

Adequate debugging support is an important prerequisite for the broad application of ontologies in real-world scenarios and in recent years, different techniques for the automated detection of problematic chunks in the knowledge bases have been developed.

One of the most critical and time-intensive tasks in most debugging approaches is the detection of small sets of axioms that contain the faults (conflict sets).

In general, efficient conflict computation and minimization is central not only in debugging scenarios, as conflict sets are also helpful to compute justifications for axioms and assertions which in turn can serve as a basis of an explanation facility [8].

In this paper we have analyzed two recent proposals for the identification of conflicts, both in black-box and glass-box application scenarios. Both the theoretical analysis as well as an empirical evaluation showed that QUICKXPLAIN is currently the more efficient method for that purposes.

Due to the lack of publicly available mass data about typical ontology faults, artificial tests had to be used in the experiments. For future work, it would be useful, if ontology editors like Protégé or Swoop would support anonymous user feedback for debugging purposes, as statistics on the number of conflict sets and their average cardinality. This data would help to make even more precise evaluations of the average case.

Finally, note that in the context of this work we generally understand axioms as valid description logics statements of any kind. Concept definitions where the left-hand sides are atomic, are the most frequent form of axioms, since available ontology editors support mainly this presentation. However, if the terminology includes axioms with a different structure, approaches like those presented in [3, 7] can be used to transform the axioms. These approaches support fine-grained debugging of ontologies, will allow us to locate faults within parts of the original axioms. Although the evaluation in this paper was limited to the more coarse-grained case, the conflict computation techniques can be applied also for the fine-grained debugging approaches.

REFERENCES

[1] *The Description Logic Handbook: Theory, Implementation, and Applications*, eds., Franz Baader, Diego Calvanese, Deborah L. McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, Cambridge University Press, 2003.

[2] Mathieu d'Aquin, Claudio Baldassarre, Laurian Gridinoc, Sofia Angeletou, Marta Sabou, and Enrico Motta, 'Watson: A gateway for next generation semantic web applications', in *Poster session of the International Semantic Web Conference, ISWC*, (2007).

[3] Gerhard Friedrich, Stefan Rass, and Kostyantyn Shchekotykhin, 'A general method for diagnosing axioms', in *DX'06 - 17th International Workshop on Principles of Diagnosis*, eds., C.A. Gonzalez, T. Escobet, and B. Pulido, pp. pp. 101–108, Penaranda de Duero, Burgos, Spain, (2006).

[4] Gerhard Friedrich and Kostyantyn Shchekotykhin, 'A General Diagnosis Method for Ontologies', *Proceedings of the 4 thInternational Semantic Web Conference (ISWC-05)*, 232–246, (2005).

[5] Russell Greiner, Barbara A. Smith, and Ralph W. Wilkerson, 'A correction to the algorithm in Reiter's theory of diagnosis', *Artificial Intelligence*, **41**(1), 79–88, (1989).

[6] Ulrich Junker, 'QUICKXPLAIN: Preferred explanations and relaxations for over-constrained problems.', in *Association for the Advancement of Artificial Intelligence*, pp. 167–172, San Jose, CA, USA, (2004).

[7] Aditya Kalyanpur, *Debugging and repair of OWL ontologies*, Ph.D. dissertation, University of Maryland, College Park, MD, USA, 2006. Adviser-James Hendler.

[8] Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin, 'Finding all justifications of OWL DL entailments', in *Proc. of ISWC/ASWC2007, Busan, South Korea*, volume 4825 of *LNCS*, pp. 267–280, Berlin, Heidelberg, (November 2007). Springer Verlag.

[9] Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler, 'Debugging unsatisfiable classes in OWL ontologies', *Web Semantics: Science, Services and Agents on the World Wide Web*, **3**(4), 268–293, (2005).

[10] Raymond Reiter, 'A theory of diagnosis from first principles', *Artificial Intelligence*, **23**(1), 57–95, (1987).

[11] Stefan Schlobach, 'Diagnosing terminologies.', in *Proc of AAAI*, eds., Manuela M. Veloso and Subbarao Kambhampati, pp. 670–675. AAAI Press / The MIT Press, (2005).

[12] Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank Harmelen, 'Debugging incoherent terminologies', *J. Autom. Reason.*, **39**(3), 317–349, (2007).

[13] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur, and Yarden Katz, 'Pellet: A practical OWL-DL reasoner', Technical report, UMIACS Technical Report, (2005).

[14] H. Wang, M. Horridge, A. Rector, N. Drummond, and J. Seidenberg, 'Debugging OWL-DL Ontologies: A Heuristic Approach', *Proceedings of the 4 thInternational Semantic Web Conference (ISWC-05)*, 745–757, (2005).