A Relevance-based Algorithm for Finding Justifications of DL Entailments*

Qiu Ji, Guilin Qi, Peter Haase

AIFB Institute
University of Karlsruhe
D-76128 Karlsruhe, Germany
{qiji,gqi,pha}@aifb.uni-karlsruhe.de

Abstract. Finding the justifications of an entailment, i.e. minimal sets of axioms responsible for a particular entailment, is an important problem in ontology engineering and thus has become a key reasoning task for Description Logic-based ontologies. While algorithms for finding all justifications of an entailment exist, key problems are efficiency and the ability to focus on *relevant* justifications. To this end, we propose a novel black-box algorithm to find justifications of an entailment using a relevance-based *selection function*. The algorithm iteratively constructs a set of justifications that are as relevant to the entailment as possible. Furthermore, each justification returned by our algorithm is attached with a weight denoting its relevance degree w.r.t. the entailment. Finally, we present evaluation results that show the benefits of the algorithm with real-life ontologies.

1 Introduction

Ontologies play a central role for the formal representation of knowledge on the Semantic Web. In logic based ontology languages, such as Description Logics (DLs), finding the justifications of an entailment is an important problem that has many practical applications, such as handling inconsistency in an ontology [8, 13] and diagnosing terminologies [11].

Several methods have been proposed to find the justifications for an entailment (see [12, 7, 3]). Although practical techniques to find all possible justifications exist, efficiency is still a problem (see [13]: As shown in [3], in the worst case the number of justifications for a subsumption entailment is exponential in the size of the ontology. Yet, in practice it is rarely required to find *all* justifications. As an illustration, according to the statistics given in [7], for unsatisfiability entailments in the real-life ontology Chemical ontology there exist up to 26 justifications, and a single justification contains up to 12 axioms. In such cases, understanding all the justifications is a tedious effort for a human. Instead of a guarantee to find all justifications, the user is typically only interested in a set of *relevant* justifications. At the same time, the restriction to only find relevant justifications allows for considerable improvements in efficiency.

To this end, we propose a novel algorithm for finding justifications of a DL-based entailment by using a relevance-based selection function. Our algorithm is based on

* This work is partially supported by the EU under the IST project NeOn (IST-2006-027595, http://www.neon-project.org/).

a black-box approach, and thus it can be implemented using any DL reasoner. More specifically, we consider finding justifications for a particular type of entailment: concept unsatisfiability. This does not restrict the generality of our approach, as for expressive description logics like $\mathcal{SHOIN}(\mathcal{D})$ (the description logic underlying OWL DL), any entailment can be reduced to concept unsatisfiability [5]. We first define a relevance-based ordering on the justifications, which allows us to associate a relevance degree with each of the justifications to facilitate the comparison among them. Specifically, we use a syntactic selection function based on concept relevance, whose intuition is to select axioms that are closely connected to the unsatisfiable concept. We then present an algorithm to find a set of justifications for an unsatisfiable concept. The algorithm incrementally selects subsets of the ontology using the selection function and finds a set of justifications from these sub-ontologies for the concept. When computing justifications from a sub-ontology, our algorithm allows for different strategies: It either computes a set of justifications that satisfies some condition(s) or computes all justifications. The algorithm is implemented and evaluated using the KAON2 reasoner[1] for reasoning tasks. We also evaluate the efficiency and effectiveness of our algorithm using some real ontologies as data sets. By comparing our algorithm and the algorithm given in [7], we show the advantage of introducing the selection function to find justifications incrementally.

The paper is organized as follows. We first briefly introduce *Description Logics* and *Selection Functions* in Section 2. In Section 3, related work is discussed. After that, we present our relevance-based algorithm in Section 4. Experimental Results are reported in Section 5. Section 6 summarizes the conclusions and discusses future work.

2 Preliminaries

2.1 Justification in Description Logics

We presume that the reader is familiar with Description Logics (DLs) and refer to the DL handbook [1] for more details. A DL-based ontology (or knowledge base) $O = (\mathcal{T}, \mathcal{R}, \mathcal{A})$ consists of a set $\mathcal{T}$ of concept axioms (TBox), a set $\mathcal{R}$ of role axioms (RBox), and a set $\mathcal{A}$ of assertional axioms (ABox). Concept axioms (or terminology axioms) have the form $C \sqsubseteq D$ where C and D are (possibly complex) concept descriptions, and role axioms are expressions of the form $R \sqsubseteq S$, where R and S are (possibly complex) role descriptions. The ABox contains *concept assertions* of the form $C(a)$ where C is a concept and a is an individual name, and *role assertions* of the form $R(a, b)$, where R is a role and a and b are individual names.

An interpretation $\mathcal{I} = (\triangle^{\mathcal{I}}, \cdot^{\mathcal{I}})$ consists of a non-empty domain set $\triangle^{\mathcal{I}}$ and an interpretation function $\cdot^{\mathcal{I}}$, which maps from individuals, concepts and roles to elements of the domain, subsets of the domain and binary relations on the domain, respectively. Given an interpretation $\mathcal{I}$, we say that $\mathcal{I}$ satisfies a concept axiom $C \sqsubseteq D$ (resp., a role inclusion axiom $R \sqsubseteq S$) if $C^{\mathcal{I}} \subseteq D^{\mathcal{I}}$ (resp., $R^{\mathcal{I}} \subseteq S^{\mathcal{I}}$). Furthermore, $\mathcal{I}$ satisfies a concept assertion $C(a)$ (resp., a role assertion $R(a, b)$) if $a^{\mathcal{I}} \in C^{\mathcal{I}}$ (resp., $(a^{\mathcal{I}}, b^{\mathcal{I}}) \in R^{\mathcal{I}}$). An interpretation $\mathcal{I}$ is called a *model* of an ontology O, iff it satisfies each axiom in O.

[1] http://kaon2.semanticweb.org/

A concept C in an ontology O is unsatisfiable if for each model $\mathcal{I}$ of O, $C^{\mathcal{I}} = \emptyset$. An ontology O is incoherent if there exists an unsatisfiable concept in O.

We introduce the notion of a justification defined in [7].

Definition 1. *(JUSTIFICATION) Let O be a consistent DL-based ontology and ax an axiom such that $O \models ax$. A subset $O' \subseteq O$ is a justification for ax in O, if $O' \models ax$, and $O'' \not\models ax$ for every $O'' \subset O'$.*

Informally, a justification for an entailment is a minimal subset of the axioms in the ontology responsible for the entailment. A particular type of entailment in a DL is concept unsatisfiability entailment, i.e., $O \models C \sqsubseteq \bot$ for concept C and bottom concept $\bot$. A justification for a concept unsatisfiability entailment is also called a justification for an unsatisfiable concept or *minimal unsatisfiability-preserving sub-ontology (MUPS)* [12].

It is well-known that for some very expressive DLs such as $\mathcal{SHOIN}(\mathcal{D})$, for every axiom ax entailed by an ontology O, there is always a concept C_{ax} that is unsatisfiable w.r.t. O [5]. For any DL that contains at least DL $\mathcal{ALC}$, the axioms responsible for the concept subsumption entailment $O \models A \sqsubseteq B$ are precisely those responsible for the concept unsatisfiability entailment $O \models A \sqcap \neg B \sqsubseteq \bot$. Therefore, in this paper, we focus on finding justifications for an unsatisfiable concept. We use ALL_JUST(C, O) to denote the set of all the justifications for an unsatisfiable concept.

2.2 Selection Functions

We introduce the notion of a selection function in a single ontology given in [6].

Definition 2. *(Selection Function) Let $\mathbf{L}$ be an ontology language (denoting sets of axioms), a selection function for $\mathbf{L}$ is a mapping $s_{\mathbf{L}}$: $\mathcal{P}(\mathbf{L}) \times \mathbf{L} \times \mathbb{N} \rightarrow \mathcal{P}(\mathbf{L})$ such that $s_{\mathbf{L}}(O, \phi, k) \subseteq O$, where $\mathcal{P}(\mathbf{L})$ is the power set of $\mathbf{L}$.*

That is, a selection function selects a subset of an ontology w.r.t. an axiom at step k. A syntactic relevance-based selection function is given as follows.

Let ϕ be an axiom in a DL-based ontology. We use $I(\phi)$, $C(\phi)$ and $R(\phi)$ to denote the sets of individual names, concept names, and role names appearing in ϕ respectively.

We first define the direct relevance between two axioms.

Definition 3. *Two axioms ϕ and ψ are* directly relevant *iff there is a common name which appears both in ϕ and ψ, i.e., $I(\phi) \cap I(\psi) \neq \emptyset$ or $C(\phi) \cap C(\psi) \neq \emptyset$ or $R(\phi) \cap R(\psi) \neq \emptyset$.*

Based on the notion of direct relevance, we can define the notion of relevance between an axiom and an ontology.

Definition 4. *An axiom ϕ is relevant to an ontology O iff there exists an axiom ψ in O such that ϕ and ψ are directly relevant.*

We introduce the relevance-based selection function which can be used to find all the axioms in an ontology that are relevant to an axiom to some degree.

Definition 5. *Let O be an ontology, ϕ be an axiom and k be an integer. The* relevance-based selection function*, written s_{rel}, is defined inductively as follows:*
$s_{rel}(O, \phi, 0) = \emptyset$
$s_{rel}(O, \phi, 1) = \{\psi \in O : \phi \mathit{\ and\ } \psi \mathit{\ are\ directly\ relevant}\}$
$s_{rel}(O, \phi, k) = \{\psi \in O : \psi$ *is directly relevant to* $s_{rel}(O, \phi, k-1)\}$*, where* $k > 1$.

We call $s_{rel}(O, \phi, k)$ the k-relevant subset of O w.r.t. ϕ. For convenience, we define $s_k(O, \phi) = s_{rel}(O, \phi, k) \setminus s_{rel}(O, \phi, k-1)$ for $k \geq 1$.

To apply the relevance-based selection function to a concept, we need to construct a new axiom which states that this concept is sub-concept of a fresh concept which does not appear in the ontology. For notational simplicity, we use $s_{rel}(O, C, k)$ to denote $s_{rel}(O, C \sqsubseteq D, k)$ where D is a fresh concept. Similarly, $s_k(O, C)$ indicates $s_k(O, C \sqsubseteq D)$.

Example 1. Consider the following ontology (taken from the Proton ontology, c.f. experiments in Section 5).

O = { Manager ⊑ Employee, Employee ⊑ JobPosition, Leader ⊑ JobPosition, JobPosition ⊑ Situation, Situation ⊑ Happening, Leader ⊑ ¬ Patent, Happening ⊑ ¬ Manager, JobPosition ⊑ ¬ Employee, JobPosition(lectureship) }.

We have

$s_1(O, Manager)$ = { Manager ⊑ Employee, Happening ⊑ ¬ Manager }

$s_2(O, Manager)$ = { Employee ⊑ JobPosition, JobPosition ⊑ ¬ Employee, Situation ⊑ Happening }

$s_3(O, Manager)$ = { Leader ⊑ JobPosition, JobPosition ⊑ Situation, JobPosition(lectureship) }

$s_4(O, Manager)$ = { Leader ⊑ ¬ Patent }

2.3 Hitting Set Tree Algorithm

We briefly introduce some notions Reiter's Hitting Set Tree algorithm given in [10] which will be used in our algorithm. We follow the reformulated notions in Reiter's theory given in [7]. Given a *universal set* U, and a set $S = \{s_1, ..., s_n\}$ of subsets of U which are *conflict sets*, i.e. subsets of the system components responsible for the error. A *hitting set* T for S is a subset of U such that $s_i \cap T \neq \emptyset$ for all $1 \leq i \leq n$. A *minimal hitting set* T for S is a hitting set such that no $T' \subset T$ is a hitting set for S. A hitting set T is cardinality-minimal if there is no other hitting set T' such that $|T'| < |T|$. Reiter's algorithm is used to calculate minimal hitting sets for a collection $S = \{s_1, ..., s_n\}$ of sets by constructing a labeled tree, called a Hitting Set Tree (HST). In a HST, each node is labeled with a set $s_i \in S$, and each edge is labeled with an element in $\cup_{s_i \in S} s_i$. For each node n in a HST, let $H(n)$ be the set of edge labels on the path from the root of the HST to n. Then the label for n is any set $s \in S$ such that $s \cap H(n) = \emptyset$, if such a set exists. Suppose s is the label of a node n, then for each $\sigma \in s$, n has a successor n_σ connected to n by an edge with σ in its label. If the label of n is the empty set, then we have that $H(n)$ is a hitting set of S. In the case of finding justifications, the universal set corresponds to ontology and a conflict set corresponds to a justification [7].

3 Related Work

The first work on finding the justifications for an unsatisfiable concept is reported in [12] where the notion of MUPS is introduced to explain logical incoherence. The authors also provide a tableau algorithm for calculating all the justifications for concept unsatisfiability in *unfoldable* DL $\mathcal{ALC}$. This algorithm is called a top-down algorithm and is also reported in [13]. Since it is based on modifying the internals of a DL reasoner, it can be only implemented by a tableau-based reasoner. Therefore, also a black-box algorithm is proposed to calculate the justifications for concept unsatisfiability with the support of an external DL reasoner in [13]. Like our algorithm, their algorithm is also based on a relevance-based selection function. A shortcoming of their algorithm however is that the obtained justifications are usually not enough to resolve the unsatisfiability, i.e., removing one axiom from each of the justifications for the concept cannot make it satisfiable. In contrast, our algorithm allows two strategies for computing justifications, and both strategies will result in set of justifications that can be used to construct a *hitting set* for the entailment, i.e., removing axioms in this set from the ontology will invalidate the entailment. Furthermore, our algorithm computes the relevance degree of each returned justification to facilitate comparison among them.

In [8], two algorithms are proposed to find all the justifications for an unsatisfiable concept. The first algorithm, called a glass-box tracing algorithm, is an extension of the tableau algorithm for expressive DL $\mathcal{SHIF}(\mathcal{D})$. The other one is a black-box algorithm which focuses on detecting dependencies between unsatisfiable concepts. The glass-box tracing algorithm is extended to approximately cover $\mathcal{SHOIN}(\mathcal{D})$ in [7] to compute a single justification of an unsatisfiable concept. A black-box algorithm is also proposed to compute a single justification. This algorithm also uses a relevance-based selection function to "expand" a freshly generated ontology to find a superset of a single justification. Based on the algorithm for computing a single justification, a black box algorithm using classical Hitting Set Tree (HST) algorithm is given to calculate all the justifications. When choosing the strategy for computing all justifications, our algorithm can also calculate all the justifications by modifying the hitting set algorithm. Furthermore, our algorithm allows another strategy for computing a subset of all the justifications that can be used to construct some potentially minimal *hitting sets*. The key difference between our algorithm and the black-box algorithm given in [7] is that our algorithm (for both strategies) iteratively calculates a set of justifications that are as relevant to the concept as possible.

There are some other related methods. A heuristic method for finding justifications for concept unsatisfiability is given in [15]. The limitation of this method is that it can determine neither all the justifications nor enough justifications for explaining an unsatisfiable concept in general, because it is based on heuristics and pattern matching. An algorithm is given in [9] to find fine-grained justifications for an unsatisfiable concept by adapting the tableau algorithm for DL $\mathcal{ALC}$. That is, their algorithm can pinpoint both the axioms and the parts of axioms that are responsible for the unsatisfiability. Their algorithm cannot be applied to DLs that are more expressive than $\mathcal{ALC}$. In [2], the authors develop a general approach for extending a tableau-based algorithm to a pinpointing algorithm that can be used to find all the justifications for an unsatisfiable concept. Baader et.al. also present an algorithm to calculate all the justifications for a subsumption relation in the less expressive DL $\mathcal{EL}^+$ in [3]. The authors also show that finding out whether there is a justification within a given cardinality bound is an NP-complete problem and provide a practical algorithm that computes one (small, but not necessarily minimal) subset that has a given subsumption relation as consequence.

4 A Relevance-based Algorithm for Finding Justifications

In this section, we first introduce a relevance-based ordering on justifications and then present an algorithm that computes a set of justifications attached with relevance degrees.

4.1 Relevance-based Ordering on Justifications

We define an ordering on justifications using the relevance-based selection function.

Definition 6. *Let O be an ontology and C be an unsatisfiable concept in O. s_{rel} is the relevance-based selection function given by Definition 5. A relevance-based ordering on the set of all the justifications for C, written $\preceq_{rel,C}$, is defined as follows: for any two justifications J_1 and J_2 for C,*

$$J_1 \preceq_{rel,C} J_2 \quad iff \quad d_{rel,C}(J_1) \geq d_{rel,C}(J_2)$$

where, $d_{rel,C}(J_i) = max\{k : J_i \cap s_k(O,C) \neq \emptyset\}$ is used to measure the relevance degree of J_i w.r.t. C.

That is, justification J_1 is less relevant to C than J_2 if and only if the element in J_1 which is furthest from C is less relevant to that in J_2 which is furthest from C.

Example 2. (Example 1 Continued) Concept $Manager$ has the following two justifications:

(1) $J_1 = \{$ Manager $\sqsubseteq$ Employee, Employee $\sqsubseteq$ JobPosition, JobPosition $\sqsubseteq \neg$ Employee $\}$

(2) $J_2 = \{$ Manager $\sqsubseteq$ Employee, Employee $\sqsubseteq$ JobPosition, JobPosition $\sqsubseteq$ Situation, Situation $\sqsubseteq$ Happening, Happening $\sqsubseteq \neg$ Manager $\}$.

By Example 1, we have $d_{rel,Manager}(J_1) = 2$ and $d_{rel,Manager}(J_2) = 3$. Therefore, $J_2 \preceq_{rel,Manager} J_1$. It is clear that J_1 is much easier to understand than J_2.

Let us consider an important property of the relevance-based ordering:

Proposition 1. *Let O be an ontology and C be an unsatisfiable concept in O. Given a justification J for C, suppose $k = d_{rel,C}(J)$, then $J \cap s_j(O,C) \neq \emptyset$ for any $0 < j \leq k$.*

Proof. It is clear that $J \cap s_k(O,C) \neq \emptyset$. Suppose that $J \cap s_j(O,C) = \emptyset$ for some $j < k$. Then for any axiom ϕ in $s_{rel}(O,C,j-1) \cap J$ and any axiom ψ in $J \setminus s_{rel}(O,C,j-1)$, ϕ and ψ are not directly relevant because ϕ can be only directly relevant to axioms in $s_{rel}(O,C,j-1)$ or axioms in $s_j(O,C)$. We show that J cannot be a justification. Let us construct an ontology O' which contains exact those elements of J. According to [8], any justification for an unsatisfiable concept must be generated by a black-box algorithm for finding a single justification. Therefore J must be generated by the black-box algorithm. In the black-box algorithm, relevance-based selection function s_{rel} is applied to select axioms from O'. It is clear that the algorithm will select axioms from $s_1(O,C) \cap O'$, $s_2(O,C) \cap O'$, etc. However, it will stop selecting axiom when it reaches $s_{j-1}(O,C) \cap O'$ because there is no element in $s_j(O,C) \cap O'$. So the justifications found by the algorithm will not include any axiom in $J \setminus s_{rel}(O,C,j-1)$. Contradiction.

The Proposition 1 states that if the relevance degree of a justification for a concept is k, then it contains at least one axiom in the ontology that is relevant to the concept with degree j for any $0 < j < k$.

Corollary 1. *Let O be an ontology and C be an unsatisfiable concept in O. Given two justifications J_1 and J_2 for C, suppose $J_1 \preceq_{rel,C} J_2$, if $J_2 \cap s_k(O,C) \neq \emptyset$ then $J_1 \cap s_k(O,C) \neq \emptyset$ for all k, but not vice versa.*

Corollary 1 follows from Proposition 1. According to Corollary 1, suppose justification J_1 is less relevant to C than J_2, if J_2 is k-relevant to C, then J_1 must be k-relevant to C as well, where a justification is k-relevant to C if and only if it has non-empty intersection with $s_k(O,C)$. Proposition 1 and Corollary 1 together tell us that if we want to find justifications for a concept in an ontology that are more relevant to the concept, we should select those axioms in the ontology that are more relevant to the concept.

4.2 Relevance-based Algorithm for Finding Justifications

Our algorithm receives an ontology O, an unsatisfiable concept C in O and a strategy for computing justifications as inputs, and outputs a set of weighted justifications $\overrightarrow{\mathcal{J}}$ and a set of hitting sets HS for All_JUST(C,O), which we call global hitting sets. In our algorithm, we consider two strategies when expanding the hitting set tree by invoking Algorithm 2: (1) All_Just_Relevance is to compute all the justifications when we expand the HST by using Algorithm 2, and (2) CM_Just_Relevance computes those justifications that are in nodes of the cardinality-minimal hitting sets in the local HST[2] expanded by Algorithm 2.

First of all, we find the first k such that C is unsatisfiable in the k-relevant subset O' of O, i.e., the "if" condition in line 19 is satisfied. We then call Algorithm 2 to find a set of justifications for C in O' and a set of local hitting sets for ALL_JUST(C,O'). After that, we associate a relevance degree to each found justification in line 21.

We then add to O' axioms in O that are directly relevant to O'. Since HS_{local} is not empty, the "if" condition in line 7 is always satisfied. In the first "for" loop, we get all the hitting sets that are global hitting sets, i.e., C becomes satisfiable if we remove axioms in such a hitting set from O (lines 8-10). We exclude from HS_{local} those hitting sets that have been selected (line 11). Now if we choose the strategy CM_Just_Relevance and there exists a global hitting set in HS, then the algorithm terminates and returns

[2] We call the HST constructed using Algorithm 2 as a local HST, and the hitting sets found in this algorithm as local hitting sets.

Algorithm 1: REL_JUSTS(C,O,$compute\_justs\_strategy$)

```
Data: An ontology $O$ and an unsatisfiable concept $C$ of $O$, and the strategy
      $compute\_justs\_strategy$ to compute justifications.
Result: A set of weighted justifications $\overrightarrow{\mathcal{J}}$ and a set of global hitting sets $HS$
1  begin
2      Globals : $strategy \leftarrow compute\_justs\_strategy$; $\overrightarrow{\mathcal{J}} \leftarrow \emptyset$
3      $O' \leftarrow HS \leftarrow HS_{local} \leftarrow \emptyset$; $k \leftarrow 1$
4      $\mathcal{S}_{rel} \leftarrow s_k(O, C)$
5      while $\mathcal{S}_{rel} \neq \emptyset$ do
6          $O' \leftarrow O' \cup \mathcal{S}_{rel}$
7          if $HS_{local} \neq \emptyset$ then
8              for $P \in HS_{local}$ do            /* Get global hitting sets */
9                  if $C$ is satisfiable in $O \setminus P$ then
10                     $HS \leftarrow HS \cup \{P\}$
11             $HS_{local} \leftarrow \{P | P \in HS_{local}$ and $P \notin HS\}$
12             if ($strategy$ ≠All_Just_Relevance and $HS \neq \emptyset$) or ($HS_{local} = \emptyset$) then
13                 return $(\overrightarrow{\mathcal{J}}, HS)$                /* Early termination */
14             $HS_{temp} \leftarrow HS_{local}$
15             for $P \in HS_{temp}$ do            /* Expand hitting set tree */
16                 $(\mathcal{J}, HS'_{local}) \leftarrow$ COMPUTE_JUSTS$(C, O' \setminus P)$
17                 $\overrightarrow{\mathcal{J}} \leftarrow \overrightarrow{\mathcal{J}} \cup \{(J, k) | J \in$ newly found justifications$\}$
18                 $HS_{local} \leftarrow HS_{local} \cup \{P \cup P' | P' \in HS'_{local}\} \setminus \{P\}$
19         else if $C$ is unsatisfiable in $O'$ then
20             $(\mathcal{J}, HS_{local}) \leftarrow$ COMPUTE_JUSTS$(C, O')$
21             $\overrightarrow{\mathcal{J}} \leftarrow \{(J, k) | J \in \mathcal{J}\}$      /* Associate relevance degree */
22         $k \leftarrow k + 1$
23         $\mathcal{S}_{rel} \leftarrow s_k(O, C)$
24     return $(\overrightarrow{\mathcal{J}}, HS)$
25 end
```

the found weighted justifications and global hitting sets. Alternatively, if HS_{local} is empty, then there is no hitting set tree to expand anymore. So the algorithm terminates and returns the found weighted justifications and hitting sets (lines 12-13). If none of the conditions in line 12 is satisfied, we expand the hitting set tree in lines 15-18. For each local hitting set P that has been found before, we call Algorithm 2 to find a set of justifications for C in $O' \setminus P$ and a set of local hitting sets for ALL_JUST$(C, O' \setminus P)$ (line 16). After that, we update the set of weighted justifications and the set of local hitting sets (lines 17-18).

In Algorithm 2, we first find a single justification (line 3) and add it to $\mathcal{J}$ which contains all the justifications generated by the algorithm (line 4). We then create all the possible branches from the node corresponding to the new found justification (lines 5-6). In the "while" loop, we first find all the branches that do not need to expand and all the branches to be expanded (steps 9-14). Our algorithm terminates and returns the

Algorithm 2: COMPUTE_JUSTS(C,O)

```
Data: An ontology $O$ and an unsatisfiable concept $C$ of $O$
Result: A set of justifications $\mathcal{J}$ for $C$ in $O$ and a set of hitting sets $HS$
1  begin
2      $HS \leftarrow HS_1 \leftarrow \emptyset$
3      $J \leftarrow$ SINGLE_JUST$(C, O)$
4      $\mathcal{J} \leftarrow \mathcal{J} \cup \{J\}$
5      for $a \in J$ do                       /* Create all possible branches. */
6          $HS_1 \leftarrow HS_1 \cup \{\{a\}\}$
7      while true do
8          $HS_2 \leftarrow \emptyset$
9          for $(P \in HS_1)$ do
10             if $C$ is satisfiable in $O \setminus P$ then        /* Local hitting sets */
11                 $HS \leftarrow HS \cup \{P\}$
12             else                         /* Branches need to be expanded */
13                 $HS_2 \leftarrow HS_2 \cup \{P\}$
14
15         if (strategy $\neq$All_Just_Relevance and $HS \neq \emptyset$) or ($HS_1 = \emptyset$) or ($HS_2 = \emptyset$)
           then
16             return $(\mathcal{J}, HS)$
17         $HS_1 \leftarrow \emptyset$
18         for $P \in HS_2$ do
19             $J \leftarrow$ SINGLE_JUST$(C, O \setminus P)$
20             $\mathcal{J} \leftarrow \mathcal{J} \cup \{J\}$
21             for $a \in J$ do
22                 $HS_1 \leftarrow HS_1 \cup \{P \cup \{a\}\}$
23 end
```

found justifications and hitting sets if any of the following condition holds: (1) The strategy CM_Just_Relevance is chosen and we have found a local hitting set, (2) the input concept C is satisfiable in the input ontology (i.e. $HS_1 = \emptyset$ in the first iteration of "while" loop), and (3) there is no branch to expand (i.e. $HS_2 = \emptyset$). If none of the conditions is satisfied, then we expand the branches stored in HS_2. For each $P \in HS_2$, we first find a single justification for C in $O \setminus P$ and add it to $\mathcal{J}$ (steps 19 and 20). We then create all the possible new branches (lines 21-22) and go to another iteration of "while" loop.

To compute a single justification, we can take the methods given in [7]. For example, SINGLE_JUST(C, O) in Algorithm 2 can be the black-box algorithm in [7]. It first expands a freshly generated ontology O' which is a superset of a justification using a relevance-based selection function. Then O' is pruned to find the final justification, where a window-based pruning strategy is used to minimize the number of satisfiability-check calls to the reasoner.

We show the correctness of Algorithm 1 when the strategy All_Just_Relevance is chosen, i.e., our algorithm finds all the justifications for ALL_JUST(C, O) when it terminates.

Theorem 1. *Let the strategy to compute justifications in Algorithm 1 be to compute all justifications ($compute\_justs\_strategy = all\_Justification$). Suppose $\overrightarrow{\mathcal{J}}$ is the set of weighted justifications returned by REL_JUSTS(C, O, $compute\_justs\_strategy$), then $ALL\_JUST(C, O) = \{J | \exists k, (J, k) \in \overrightarrow{\mathcal{J}}\}$. For each $(J, k) \in \overrightarrow{\mathcal{J}}$, we have $k = d_{rel,C}(\mathcal{J})$.*

Proof. (sketch) It is easy to check that Algorithm 1 will terminate and return $\overrightarrow{\mathcal{J}} = \emptyset$ when C is satisfiable in O. So we assume that C is unsatisfiable in O. Since it has been shown that HST algorithm can be used to find all the justifications in [7], we only need to show that Algorithm 1 expand all the branches of the hitting set tree for ALL_JUST(C, O). This can be seen from the following facts: (1) for each local hitting set in HS_{local}, if it is a global hitting set, then it is stored in the set HS (lines 8-10); (2) Early termination will not happen unless there is no local hitting set to expand, i.e., we have found all the global hitting sets; (3) in Algorithm 2, since we have chosen the strategy All_Just_Relevance, the algorithm terminates only if there is no more branch that can be expanded, i.e., either $HS_1 = \emptyset$ or $HS_2 = \emptyset$. Since we add all the possible branches to HS_1 (lines 5-7 and lines 22-24), all the branches have been explored when the algorithm terminates. It is clear that $k = d_{rel,C}(\mathcal{J})$ according to steps 17 and 21 of Algorithm 1.

We show the correctness of Algorithm 2 when the strategy CM_Just_Relevance is chosen, i.e., it computes those justifications that are nodes of the cardinality-minimal hitting sets in the local hitting set tree expanded by Algorithm 2.

Theorem 2. *Suppose strategy CM_Just_Relevance is an input of Algorithm 1. Suppose $\mathcal{J}$ and HS are returned by COMPUTE_JUSTS(C,O), then for each local hitting set $J \in \mathcal{J}$, there is no other local hitting set J' in ALL_JUST(C, O) such that $|J'| < |J|$.*

Proof. Since we choose strategy CM_Just_Relevance, there are three cases where the algorithm terminates (lines 15-16). Case 1: $HS \neq \emptyset$, i.e., there is a hitting set in ALL_JUST(C, O). In this case, all the hitting sets in HS has the same cardinality l. It is impossible to find a hitting set whose cardinality is less than l. Otherwise, since we expand the hitting set in the breadth-first manner, this hitting set must have been put in HS before and the algorithm should have terminated. This is a contradiction. Case 2: $HS_1 = \emptyset$. In this case, we must have that there is no justification for C in O, so All_Just(C, O)=$\emptyset = \mathcal{J}$. Case 3: $HS_2 = \emptyset$. In this case, we have expanded all the branches and we must have found a hitting set, i.e., condition $HS \neq \emptyset$ is satisfied. So the conclusion holds.

Example 3. Given an ontology including the following axioms:

1: $U \sqsubseteq A$,	2: $U \sqsubseteq \neg A$,	3: $U \sqsubseteq C$,	4: $C \sqsubseteq \neg B$,
5: $A \sqsubseteq B$,	6: $U \sqsubseteq G$,	7: $G \sqsubseteq E$,	8: $U \sqsubseteq F$,

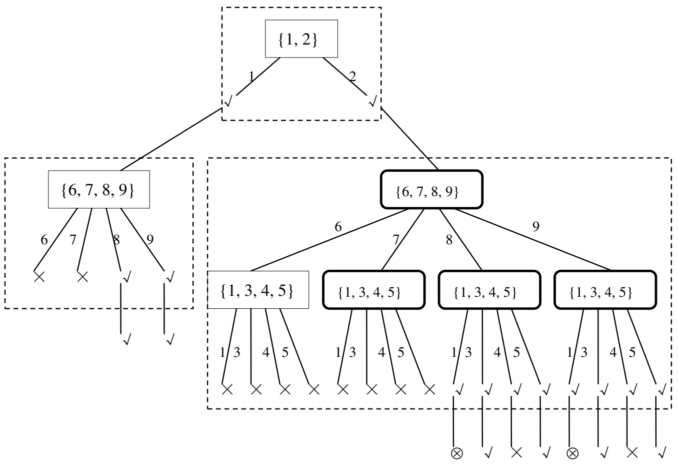

Fig. 1. Finding justifications using relevance-based algorithm: Each distinct node in a rectangular box represents a justification. And each sub-tree outlined in a rectangular box shows the process to compute justifications for a sub-ontology by invoking Algorithm 2.

9: $F \sqsubseteq \neg E$, 10: $U \sqsubseteq D$, 11: $D \sqsubseteq E$, 12: $C \sqsubseteq K$,
13: $K \sqsubseteq \neg H$, 14: $B \sqsubseteq H$

We denote each axiom by a natural number (1..14) for simplicity. In O there is only one unsatisfiable concept U. For the concept U, we obtain the following subsets of O using selection function: $s_1(O,U) = \{1,2,3,6,8,10\}$, $s_2(O,U) = \{4,5,7,9,11,12\}$ and $s_3(O,U) = \{13,14\}$. All the justifications for U are listed as follows:

$\mathcal{J} = \{\{1,2\},\{1,3,4,5\},\{6,7,8,9\},\{8,9,10,11\},\{1,3,5,12,13,14\}\}$.

We illustrate our algorithm where CM_Just_Relevance is chosen in Figure 1.

k = 1 : $O' = s_{rel}(O,U,1)$. A justification $J = \{1,2\}$ is computed by Algorithm 2 and set as the root node of the hitting set tree (HST) (see the top rectangular box in Figure 1). Then two possible branches $\{1\}$ and $\{2\}$ can be generated from J (see lines 5-7 of Algorithm 2). Since U turns satisfiable in $O' \setminus \{1\}$ and $O' \setminus \{2\}$ (see lines 10-14), we know the two branches are local hitting sets and then go back to Algorithm 1 (see lines 15-16). We then associate relevance degree 1 to each returned justification in $HS = \{\{1\},\{2\}\}$ (line 22 of Algorithm 1).

k = 2 : $O' = s_{rel}(O,U,2)$. Since no local hitting sets are global (see lines 7-10 of algorithm 1), we need to expand each branch stored in HS.

Take branch $\{2\}$ as an example (suppose we have expanded branch $\{1\}$ in the left rectangular box below the root node). We call Algorithm 2 with inputs U and $O' \setminus \{2\}$. The justification $J_1 = \{6,7,8,9\}$ which has been computed before is reused and is set as the root node of local HST for ALL_JUST($U, O' \setminus \{2\}$. Check each possible branch originated from J_1, none of them is a local hitting set and so we need to generate a new justification along each branch (see lines 18-21). For example, we generate $J_2 = \{1,3,4,5\}$ along branch $\{6\}$ and construct new branches $\{6,1\}$, $\{6,3\}$, $\{6,4\}$ and $\{6,5\}$. Similarly other branches $\{7\}$, $\{8\}$ and $\{9\}$ in $O' \setminus \{2\}$ can be generated and expanded. We then go to next iteration to check each newly generated branch and denote those local hitting sets with '$\surd$' (otherwise, with '$\times$'). From the figure we can see that 8 local hitting sets are found in $O' \setminus \{2\}$. Then we go back to Algorithm 1 and associate relevance degree 2 to the newly computed justification $\{1, 3, 4, 5\}$. Also, we replace the prefix-branch $\{2\}$ with new paths by combining the newly found local hitting sets with $\{2\}$ in line 18 of Algorithm 1.

k = 3 : $O' = s_{rel}(O, U, 3)$. In this iteration, we check all possible branches and find 8 global hitting sets which are marked with '$\surd$' outside the rectangular box. The condition in line 12 of Algorithm 1 is satisfied, so the algorithm terminates and outputs global hitting sets $HS = \{\{1, 8\}, \{1, 9\}, \{2, 8, 3\}, \{2, 8, 5\}, \{2, 9, 3\}, \{2, 9, 5\}\}$
and a set of weighted justifications $\overrightarrow{\mathcal{J}} = \{(\{1, 2\}, 1), (\{6, 7, 8, 9\}, 2), (\{1, 3, 4, 5\}, 2)\}$.

Since our relevance-based algorithm is based on modified HST algorithm [10], similar to [7], we can apply the optimized techniques in Reiter's HST algorithm in our algorithm: We can apply *justification reuse*, which means if the current edge path in any branch of the HST does not overlap with a justification, then this justification can be used as a new node to this branch. This optimization can be applied in the following ways. First, some justifications found in previous iterations may be reused. For example (see Figure 1), the justification $\{6, 7, 8, 9\}$ in oval border is reused among different sub-ontologies: one sub-ontology is $s_{rel}(O, U, 2) \setminus \{1\}$ and another one is $s_{rel}(O, U, 2) \setminus \{2\}$. Second, in one sub-ontology, it is possible to reuse some previously found justifications along different branches. For example, in the sub-ontology $s_{rel}(O, U, 2) \setminus \{2\}$, justification $\{1, 3, 4, 5\}$ is reused three times since this justification has no interaction with each branch (i.e. $\{6\}$, $\{7\}$,$\{8\}$ or $\{9\}$).

Early path termination can be applied to our algorithm in different ways. First, if the current edge path in any branch of the HST is a superset of some hitting set, this path is a hitting set as well and it is not necessary to further expand this branch. For example, the path $\{2, 8, 1\}$ marked with $\otimes$ in Figure 1 is a superset of $\{1, 8\}$ which is a previously found hitting set. Second, the current edge can be terminated if it is a prefix-path of some found hitting set. For instance, path $\{a_1, a_2\}$ is a prefix-path of a hitting set $\{a_1, a_2, a_3\}$. Finally, if strategy CM_Just_Relevance is chosen in our algorithm, once we found some local hitting sets, the other branches correspond to non-local hitting sets can be terminated. See the figure again, the branches marked with '$\times$' in each rectangular box are not expanded anymore.

5 Experiments

In this section, we present the evaluation results of our algorithm. Our algorithm was implemented in Java as part of the RaDON plugin[3] for the NeOn Toolkit [4] using KAON2 as a reasoner. To fairly compare with the debugging algorithm in [7], we re-implemented the algorithm with KAON2 API (we call it as All_Just_Alg algorithm). The experiments were performed on a Linux 2.6.16.1 System and 1024MB maximal heap space was set. Sun's Java 1.5.0 Update 6 was used for Java-based tools. For com-

[3] http://radon.ontoware.org/
[4] http://www.neon-toolkit.org/

puting justifications of a single unsatisfiable concept in each run, we set a time limit of 30 minutes.

5.1 Data Sets

Ontology	Classes	Properties	EquClass	SubClass.	DisjClass.	SubProp.	Domain	Range	Axioms
CHEM-A	48	19	18	46	6	4	18	18	114
CONFTOOL-CMT (HMatch)	68	95	1	111	70	0	95	95	457
CRS-SIGKDD (HMatch)	64	45	7	81	12	0	44	44	213
Proton	266	111	0	278	1,346	49	82	60	1,826
CONFTOOL-EKAW (HMatch)	112	69	0	189	117	8	60	60	485
EKAW-SIGKDD (HMatch)	123	61	7	204	74	8	51	51	440
KM1500	9,842	548	0	8,853	2,091	0	548	548	12,656

Table 1. Statistics of the ontologies in the data set.

Table 1 shows some characteristics of the data sets used for our experiments. *CHEM-A* was provided by Maryland University[5]. *Proton* is a basic upper-level ontology enriched with disjointness axioms[14]. *KM1500* is created by the Text2Onto ontology learning tool. The rest of data sets[6] are generated by merging ontologies and axioms resulting from a translation of the matching-results of ontology alignment system HMatch [4]. All the data sets are available for download[7].

We divide the data sets into two groups. Group one consists of four ontologies: *CHEM-A*, *CONFTOOL-CMT*, *CRS-SIGKDD*, *Proton*. For these ontologies, all the justifications of an unsatisfiable concept can be computed within the resource limits (i.e. the maximal heap space is 1024MB and the maximal time-out period is 30 minutes). Group two consists of three ontologies: *CONFTOOL-EKAW*, *EKAW-SIGKDD*, *KM1500*. For some unsatisfiable concepts in these ontologies, we cannot compute all of the justifications within the resource limits. Particularly, for ontology *KM1500*, it is impossible to compute all the justifications for most of the unsatisfiable concepts. Still, we use the ontologies to show how relevant justifications can be computed.

5.2 Evaluation Results

To evaluate the efficiency of our algorithm over data sets in group one, we compute all the justifications for all unsatisfiable concepts using All_Just_Alg and our relevance-based one with both strategies (i.e. All_Just_Relevance and CM_Just_Relevance). Figure 2 shows the average execution time for each unsatisfiable concept.[8] We observe that our algorithm is much faster than All_Just_Alg to compute the justifications in average when strategy CM_Just_Relevance is chosen. This is because fewer justifications are generated. Take the ontology *CONFTOOL-EKAW* as an example. The average number of

[5] http://www.mindswap.org/ontologies/debugging/

[6] http://webrum.uni-mannheim.de/math/lski/ontdebug/index.html

[7] http://radon.ontoware.org/downloads/datasets-iswc08.zip

[8] Please note that the results show the total time to compute the justifications, including the time to check satisfiability, unlike the results reported in [7], which excluded the time for satisfiability checking.

ontology	Strategy	# Unsatisf. Concepts	# Justifications All	# Justifications Avg	Justification_Size Avg	# Hitting sets Avg
CHEM-A	All_Just_Alg	37	412	11	9	195
	All_Just_Relevance	37	412	11	9	195
	CM_Just_Relevance	37	37	1	7	4
CONFTOOL-CMT	All_Just_Alg	26	351	14	6	191
	All_Just_Relevance	26	351	14	6	191
	CM_Just_Relevance	26	43	2	4	3
CRS-SIGKDD	All_Just_Alg	19	64	3	5	14
	All_Just_Relevance	19	64	3	5	14
	CM_Just_Relevance	19	21	1	4	3
Proton	All_Just_Alg	24	41	2	6	9
	All_Just_Relevance	24	41	2	6	9
	CM_Just_Relevance	24	24	1	6	4

Table 2. The evaluation results for data sets in group one.

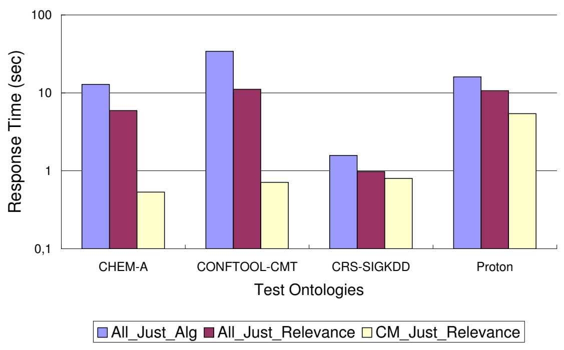

Fig. 2. The average time to compute justifications for an unsatisfiable concept.

justifications returned by All_Just_Relevance is about 14, whilst the average number is 2 when strategy CM_Just_Relevance is chosen. Another observation is that our algorithm outperforms All_Just_Alg when strategy All_Just_Relevance is chosen. It shows the advantage of incrementally increasing the size of the ontology using the selection function.

Relevance Degree k	1	2	3	4	5	6	7	8	9
s_{rel}(CONFTOOL-EKAW, Trip, k)	5	26	93	251	372	468	483	483	483
s_{rel}(EKAW-SIGKDD, Workshop_Paper, k)	5	59	173	275	368	427	435	435	435
s_{rel}(KM1500, framework, k)	57	1,335	6,678	9,730	10,892	11,206	11,305	11,336	11,342
s_{rel}(KM1500, experience, k)	48	1,286	6,675	9,856	10,897	11,218	11,307	11,335	11,342

Table 3. Number of axioms selected by s_{rel} depending on the relevance degree k

Table 3 shows the number of axioms that have been selected by the selection function s_{rel} for a given relevance degree k. We can see that sometimes the set of axioms is expanded very fast using the relevance-based selection function. For example, $|s_{rel}(O, C, 1)| = 57$, $|s_{rel}(O, C, 2)| = 1,278$ and $|s_{rel}(O, C, 3)| = 5,343$ for C=framework and O is the ontology KM1500. However, by considering the large size of ontology KM1500 (i.e. 12,656 axioms in total), it is still promising to apply the relevance-based selection function to alleviate the heavy burden to find justifications over the entire ontology. The advantage of using relevance-based algorithm can be further seen in the results below. We show the evaluation results for some unsatisfiable

(O, C)	Strategy	# of Justifications			Total # of found justifications	# Found hitting sets	Time (sec)
		$k = 1$	$k = 2$	$k = 3$			
(CONFTOOL-EKAW, Trip)	All_Just_Alg	n.a.	n.a.	n.a.	34	2,208	TO
	All_Just_Relevance	0	0	80	80	2	TO
	CM_Just_Relevance	0	0	3	3	2	3
(EKAW-SIGKDD, Workshop_Paper)	All_Just_Alg	n.a.	n.a.	n.a.	41	3,279	TO
	All_Just_Relevance	0	61	TO	61	20	TO
	CM_Just_Relevance	0	4	–	4	2	2
(KM1500, framework)	All_Just_Alg	n.a.	n.a.	n.a.	1	0	TO
	All_Just_Relevance	1	33	TO	34	2	TO
	CM_Just_Relevance	1	9	–	10	2	96
(KM1500, experience)	All_Just_Alg	n.a.	n.a.	n.a.	1	0	TO
	All_Just_Relevance	1	6	10	17	6	TO
	CM_Just_Relevance	1	–	–	1	1	6

Table 4. The evaluation results over some unsatisfiable concepts. Where "TO" means time-out (the time-out period is set to be 30 minutes) and "n.a." means "not applicable". "–" indicates the algorithm has terminated.

concepts in the data sets of group two in Table 4. The table shows the number of justifications for a given unsatisfiable concept depending on the relevance degree k (note that this does not apply to All_Just_Alg), the total number of found justifications and hitting sets, and the computation time. If the timeout was exceeded, the total number of justifications and hitting sets refers to the number found until the timeout occurred. Strategy CM_Just_Relevance performs very fast for all the selected unsatisfiable concepts. We can make the following observations: (1) For the large data set *KM1500* with more than 10 thousand axioms, only one justification is found by the All_Just_Alg within 30 minutes. This is because this algorithm performs each satisfiability check over the entire ontology, which is quite time-consuming. In contrast, our relevance-based algorithm only performs satisfiability check over relatively smaller sub-ontologies $k_{rel}(O, C, k)$ when $k \leq 3$. Take the concept "experience" as an example. Our algorithm with input strategy All_Just_Relevance returns 1, 6 and 10 justifications for $k = 1$ ($|O'| = 48$), $k = 2$ ($|O'| = 1,286$) and $k = 1$ ($|O'| = 6,675$) respectively. (2) For data sets CONFTOOL-EKAW and EKAW-SIGKDD, our algorithm with input strategy All_Just_Relevance returns more justifications than All_Just_Alg within the time limit and these justifications are most relevant to the unsatisfiable concept. (1) and (2) together show the advantage of introducing the selection function to find justifications incrementally. (3) Our algorithm with input strategy CM_Just_Relevance is much more efficient than other two (e.g. 3 seconds for concept "Trip" in *CONFTOO-EKAW*, 96 seconds for concept "framework" in *KM1500*), although much less justifications are returned.

6 Conclusion and Future Work

In this paper, we presented a novel algorithm to find justifications for DL concept unsatisfiability entailment. We first introduced a relevance-based ordering on justifications for an unsatisfiable concept and thus provided a criterion of comparison between different justifications. We then provided a novel algorithm to find justifications for an unsatisfiable concept based on a relevance-based selection function. Our algorithm allows for two different strategies when calculating justifications: All_Just_Relevance and CM_Just_Relevance. Using the first strategy, our algorithm can find all the justifications for an unsatisfiable concept. When the second strategy is chosen, our algorithm usually does not calculate all the justifications but a subset of them that satisfies some minimality condition. Based on experimental results, we showed that our algorithm is very promising compared with the HST-based algorithm in [7] for both strategies. More specifically, our algorithm is much more efficient than the HST-based one when the strategy CM_Just_Relevance is chosen. Although only a small subset of the justifications are returned by our algorithm when the strategy CM_Just_Relevance is chosen, these justifications still provide partially complete view of the unsatisfiability as they correspond to hitting sets for the set of all the justifications. While we have shown that even a simple syntax-based selection function yields useful results, for future work we will investigate more powerful selection functions, such as a semantic relevance selection function. We will also extend our algorithm by considering uncertainty.

References

1. Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter Patel-Schneider. *The Description Logic Handbook: Theory, Implementation and Application*. Cambridge University Press, 2003.
2. Franz Baader and Rafael Peñaloza. Axiom pinpointing in general tableaux. In *Proc. of TABLEAUX'07*, pages 11–27, 2007.
3. Franz Baader, Rafael Peñaloza, and Boontawee Suntisrivaraporn. Pinpointing in the description logic EL^+. In *Proc. of KI'07*, pages 52–67, 2007.
4. Silvana Castano, Alfio Ferrara, and Stefano Montanelli. Matching ontologies in open networked systems: Techniques and applications. pages 25–63, 2006.
5. Ian Horrocks and Peter F. Patel-Schneider. Reducing OWL entailment to description logic satisfiability. *J. Web Sem.*, 1(4):345–357, 2004.
6. Zhisheng Huang, Frank van Harmelen, and Annette ten Teije. Reasoning with inconsistent ontologies. In *Proc. of IJCAI'05*, pages 254–259. 2005.
7. Aditya Kalyanpur, Bijan Parsia, Matthew Horridge, and Evren Sirin. Finding all justifications of OWL DL entailments. In *Proc. of ISWC/ASWC'07*, pages 267–280, 2007.
8. Aditya Kalyanpur, Bijan Parsia, Evren Sirin, and James Hendler. Debugging unsatisfiable classes in OWL ontologies. *Journal of Web Semantics*, 3(4):268–293, 2005.
9. Joey Sik Chun Lam, Derek H. Sleeman, Jeff Z. Pan, and Wamberto Weber Vasconcelos. A fine-grained approach to resolving unsatisfiable ontologies. *Journal of Data Semantics*, 10:62–95, 2008.
10. Raymond Reiter. A theory of diagnosis from first principles. *Artificial Intelligence*, 32(1):57–95, 1987.
11. Stefan Schlobach. Diagnosing terminologies. In *Proc. of AAAI'05*, pages 670–675. AAAI Press, 2005.
12. Stefan Schlobach and Ronald Cornet. Non-standard reasoning services for the debugging of description logic terminologies. In *Proc. of IJCAI'03*, pages 355–362. 2003.
13. Stefan Schlobach, Zhisheng Huang, Ronald Cornet, and Frank van Harmelen. Debugging incoherent terminologies. *J. Autom. Reasoning*, 39(3):317–349, 2007.
14. Johanna Völker, Denny Vrandecic, York Sure, and Andreas Hotho. Learning disjointness. In *Proc. of ESWC'07*, pages 175–189, 2007.
15. Hai Wang, Matthew Horridge, Alan Rector, Nick Drummond, and Julian Seidenberg. Debugging OWL-DL ontologies: A heuristic approach. In *Proc. of ISWC'05*, pages 745–757, 2005.