Front. Comput. Sci., 2020, 14(4): 144305
https://doi.org/10.1007/s11704-019-7267-5

RESEARCH ARTICLE

Extracting a justification for OWL ontologies by critical axioms

Yuxin YE[1,2], Xianji CUI[2,3], Dantong OUYANG (✉)[1,2]

1 College of Computer Science and Technology, Jilin University, Changchun 130012, China
2 Key Lab of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, Changchun 130012, China
3 College of Information and Communication Engineering, Dalian Minzu University, Dalian 116600, China

Abstract Extracting justifications for web ontology language (OWL) ontologies is an important mission in ontology engineering. In this paper, we focus on black-box techniques which are based on ontology reasoners. Through creating a recursive expansion procedure, all elements which are called critical axioms in the justification are explored one by one. In this detection procedure, an axiom selection function is used to avoid testing irrelevant axioms. In addition, an incremental reasoning procedure has been proposed in order to substitute series of standard reasoning tests w.r.t. satisfiability. It is implemented by employing a pseudo model to detect "obvious" satisfiability directly. The experimental results show that our proposed strategy for extracting justifications for OWL ontologies by adopting incremental expansion is superior to traditional Black-box methods in terms of efficiency and performance.

Keywords description logics, automated reasoning, ontology engineering, justification

1 Introduction

Ontology acquisition and maintenance are important prerequisites for the successful application of ontology in a variety of areas, such as semantic web. There exist some solutions for assisting humans to modify ontologies. Justifications are minimal entailing subsets of an web ontology language (OWL) ontology which provide helpful and easy-to-understand explanations for repairing unwanted entailments in the ontology debugging process.

Finding justifications of an entailment (i.e., all minimal set of axioms sufficient to produce an entailment) has emerged as a key inference service for the OWL. Schlobach and Cornet were among the first to consider debugging of erroneous terminologies as a new non-standard reasoning services in [1]. They defined a concept of minimal unsatisfiability-preserving sub-TBoxes (MUPS) to explain the cause of incoherences in description logic (DL) terminologies. In fact, MUPS is a special justification which were defined in [2] by Kalyanpur et al. In their work, they also expanded the handled object from terminologies to the whole ontology. Both [2] and [3] categorized the solution strategies into two ways. One is a black-box technique which solely uses the reasoner as a subroutine and the internals of reasoner are not needed to be modified. The other one is a glass-box technique which is built on existing tableau based decision procedures for expressive DL. Their implementations required a thorough and non-trivial modification of the internals of reasoner. The new resolution method we proposed in this paper belongs to black-box technique which is independent of reasoners. There are some others strategies besides black-box or glass-box techniques, such as automata-based axiom pinpointing considered in [4, 5], and so on.

In [6], the contextual knowledge is exploited to detect the hidden modeling errors of incoherent ontologies. Xue et al. [7] proposed a novel topic hierarchy construction method for the topic classification which could also be ap-

Received July 30, 2017; accepted March 14, 2018

E-mail: ouyd@jlu.edu.cn

plied to the concept hierarchy of an ontology. For the reasoning of multi-agent systems, Wu et al. [8] proposed a quantified temporal knowledge, belief and certainty logic which were sound and complete with respect to the corresponding semantical classes. Ouyang et al. [9] provided a modified tableau algorithm which was sound and complete for deciding the consistency of an extended ontology. The work in [10] considered an optimization technique based on the unsatisfiable dependent path to calculate the minimal unsatisfiability preserving sub-TBoxes of an unsatisfiable concept, for which the optimization was on the basis of glass box method of ontology debugging. Considering the fact that ontologies in real world applications are typically dynamic entities when new knowledge are added in the existing ontologies, [11] used a heuristic strategy to reuse the results of previous debugging and to provide information for the next debugging.

Justification is not only an important explanation for erroneous ontologies, but also the infrastructure for many other ontology debugging missions. [1] defined the MIPS (i.e., minimal in-coherence preserving sub-TBox) based on MUPS. Furthermore, [12] and [13] separately constructed the diagnosis system by finding the minimal hitting set of MIPS. A few debugging strategies were also proposed for handling the problem about how to delete conflict axioms according to such diagnosis system. [14] and [15] developed the theory of ontology diagnosis in terms of interaction and parallelization. Other related works include understanding entailments in OWL [16], decluttering justifications [17], and so on.

2 Motivation

Informally, a justification is simply the precise set of axioms in an ontology responsible for a particular entailment. For example, if an ontology $\mathcal{O}$ contains concepts A, B, and A is inferred to be a subclass of B, i.e., $\mathcal{O} \models A \sqsubseteq B$, then the justification $\mathcal{J}$ for this concept subsumption entailment is the smallest set of axioms in $\mathcal{O}$ responsible for it. The concept of justifications formally defined as follows:

Definition 1 (Justification [2]) $\mathcal{J}$ is a justification for $\mathcal{O} \models \eta$, if $\mathcal{J} \subseteq \mathcal{O}$, $\mathcal{J} \models \eta$, and for all $\mathcal{J}' \subset \mathcal{J}$, it holds that $\mathcal{J}' \not\models \eta$. It's denoted as **JUST**$(\eta, \mathcal{O})$.

In this way, all justifications for η in $\mathcal{O}$ are denoted as **ALL_JUST** $(\eta, \mathcal{O})$. The most intuitive idea of finding justifications for an OWL ontology $\mathcal{O}$ is to decide whether the entailment relationship is hold between the target axiom and every sub-ontology of $\mathcal{O}$ at first. There are 2^m testing times for m axioms included in $\mathcal{O}$. All of sub-ontologies will be found after successful entailment testing. All pairs of such sub-ontologies are compared with each other to discover minimal sub-ontologies. In fact, there would be $(2^m-1)^2/2$ testing times in the worst case. Even though such procedure is sound and complete, it is exhaustive and an impossible mission for large scale ontology.

Many strategies have been proposed to solve this problem. The common idea is to first determine a certain justification, and then find all the other justifications via using Hitting-Tree algorithm [18]. In this paper, we mainly focus on how to detect one justification efficiently. According to the naive approach discussed in [2], finding the first justification only needs m testing times for an ontology including m axioms. Their method tries to decide a series of satisfaibilities for this ontology by an expansion-contraction procedure. Even if its computation complexity can be handled by programming, there are still some possible improvement. First of all, the detection of justification is only depend on the contractive phase, but not related with expansive phase. Thus one may think of if it is possible to detect the justification or a part of justification directly during the expansive phase. Secondly, although it is easily operated given every entailment testing occurred in this procedure is independent with each other, it would generate unnecessary reasoning expenses. In this paper, we adopt a new solution architecture to the extraction of a justification. Under our architecture, the justification can be detected by the expansive phase directly through defining the critical axiom. Inner relationships between entailment testings which occurred in this expansive phase has also been discovered. Consequently, the proposed new strategy can reduce the reasoning procedure thus it is able to find the justification faster.

3 Definition of critical axiom and construction of a justification

3.1 Critical axiom

As aforementioned, the existing strategy for the extration of justification is achieved by an expansion-contraction process. The extraction procedure is implemented by selecting relevant and removing irrelevant axioms from the ontology. In contrast to this expansion-contraction procedure, we here construct an iterative expansion procedure in order to avoid running contraction phases. Through a series of iterative expansions, all axioms in the same justification could be determined. We define such axiom as critical axiom in our al-

gorithm. Similar concept with respect to constraint problem or SAT problem is called critical clause in [19] or necessary clause in [20].

Definition 2 (Critical axiom) Given an axiom η, we say that an axiom α of an ontology O is a critical axiom with respect to O and η, if α is an axiom that belongs to every justification of O for the axiom η. In other words, axiom α is critical if $\alpha \in \bigcap_{i=1}^{n} \mathbf{JUST}(\eta, O)$, in that $n = |\mathbf{ALL\_JUST}(\eta, O)|$. A critical axiom is denoted as $\mathbf{CRIT}(\eta, O)$.

Furthermore, all critical axioms for η in O are denoted as $\mathbf{ALL\_CRIT}(\eta, O) = \{\alpha | \alpha \in \bigcap_{i=1}^{n} \mathbf{JUST}(\eta, O)\}$, in that $n = |\mathbf{ALL\_JUST}(\eta, O)|$.

For example, as shown in Fig. 1, O_1 is an ontology O_1 including four axioms and η_1 is an entailment axiom. Base on O_1 and η_1, it's easy to obtain two justifications as the following:

$\mathcal{J}_1 = \{\alpha_1 : A \sqsubseteq \exists r.B, \alpha_2 : Domain(r, C)\} \models A \sqsubseteq C$

$\mathcal{J}_2 = \{\alpha_3 : A \sqsubseteq \exists s.B, \alpha_4 : r \sqsubseteq s, \alpha_2 : Domain(r, C)\} \models A \sqsubseteq C$

They are both minimal axiom sets which entail the axiom η_1: $A \sqsubseteq C$. α_2 is the common axiom w.r.t all justifications. According to Definition 2, α_2 is the critical axiom w.r.t. O_1 and η_1.

ontology O_1	$\alpha_1 : A \sqsubseteq \exists r.B$	$\alpha_2 : Domain(r, C)$
	$\alpha_3 : A \sqsubseteq \exists s.B$	$\alpha_4 : r \sqsubseteq s$
axiom η_1	$\eta_1 : A \sqsubseteq C$	

Fig. 1 An ontology illustration

Usually, there is no any critical axiom for the whole ontology, because the critical axiom has to belong to every $\mathbf{JUST}(\eta, O)$. Sometimes justifications are disjoint. For example, we use α_3' and α_4' instead of α_3 and α_4 to establish a new ontology O_1' as shown in Fig. 2.

ontology O_1'	$\alpha_1 : A \sqsubseteq \exists r.B$	$\alpha_2 : Domain(r, C)$
	$\alpha_3' : A \sqsubseteq D$	$\alpha_4' : D \sqsubseteq C$
axiom η_1	$\eta_1 : A \sqsubseteq C$	

Fig. 2 Another ontology illustration

$\mathcal{J}_1$ is still set up w.r.t. O_1' and η_1, and $\mathcal{J}_2$ is changed to be $\mathcal{J}_2'$ as the following:

$\mathcal{J}_1 = \{\alpha_1 : A \sqsubseteq \exists r.B, \alpha_2 : Domain(r, C)\} \models A \sqsubseteq C$

$\mathcal{J}_2' = \{\alpha_3' : A \sqsubseteq D, \alpha_4' : D \sqsubseteq C\} \models A \sqsubseteq C$

$\mathcal{J}_1$ and $\mathcal{J}_2'$ are disjoint given there is no common element between these two justifications. Therefore, no critical axiom exists w.r.t. O_1' and η_1. However, we can still detect a critical axiom in a small enough ontology which could be a sub-ontology of some original ontology. In our example, O_1'' is a sub-ontology of O_1' when O_1'' only includes α_1 and α_2 (similar in the scenario where O_1'' only includes α_3' and α_4'). Every axiom of O_1'' is critical w.r.t. O_1'' and η_1, but not w.r.t. O_1' and η_1.

3.2 Method for critical axiom detection

According to the definition of critical axioms, it is easy to imagine that a critical axiom for the whole ontology usually does not exist, since it is difficult to find a axiom belonging to every $\mathbf{JUST}(\eta, O)$ at the same time. However, we may try to define a critical axiom in a small enough ontology O' which is a sub-ontology of O. In the following, a function is constructed in order to detect a critical axiom by extracting a small sub-ontology from the whole ontology. Arguments of this function include ontology O, entailment axiom η, and sub-ontology O' of O. Usually, O' is initially set up as empty. Then. O' is updated by adding axioms from O until it entails η. The final updated O' and last appended axiom α (actually it's critical w.r.t. O' and η) will be the output of the function.

Function 1 Find_Critical_Axiom (O, O', η)

Input: Ontology $O, O' \subseteq O$, Entailment Axiom η;
Output: $\mathbf{CRIT}(\eta, O')$ and sub-ontology O';
1: **while** η is not entailed by O' **do**
2: select some axiom α from $O \backslash O'$
3: if η is not entailed by $O' \cup \{\alpha\}$ **then**
4: $O' \leftarrow O' \cup \{\alpha\}$
5: **end if**
6: **end while**
7: $O' \leftarrow O' \cup \{\alpha\}$
8: **return** *last appended*$(\{\alpha\})$ and O'

Assume that the sub-ontology O' of O is initialized as empty, so O' does not entail η at first. Thus the decision condition in step 1 is satisfied, and then the program can be executed into the loop body containing steps from 2 to 6. In this loop body, O' is firstly selected as any axiom from $O \backslash O'$ (step 2), then it checks whether η is entailed by the union of $\{\alpha\}$ and O'. If the condition $O' \models \eta$ does not hold (step 3), O' is updated by adding the current selected axiom α (step 4). This recursive procedure will terminate until $O' \cup \{\alpha\} \models \eta$. At the end of this loop procedure, the final ontology O' is updated by $O' \cup \{\alpha\}$ (step 6). Such final ontology O' is the sub-ontology including a critical axiom, and the last axiom tested is the critical one for O' (step 8). Proposition 1 guarantees that Function 1 is correct and sound. The axiom generated by Function 1 is absolutely critical w.r.t. η and updated O'. The

proposition can be easily proved according to Definition 2.

Proposition 1 The last axiom α appended to $\mathcal{O}'$ by Function 1 is a critical axiom w.r.t. $\mathcal{O}'$ and η.

Proof According to Function 1, $\mathcal{O}'$ does not entail η when the last axiom α is not included in $\mathcal{O}'$. At this moment, there is no justification generated based on Function 1. However at least one justification will be generated until the last axiom α being added into $\mathcal{O}'$. This implies that the last axiom α must be included in any justification w.r.t. η and the final $\mathcal{O}'$. If there is any justification uninvolved with α for η and the final $\mathcal{O}'$, it must exists in the previous $\mathcal{O}'$, which is generated before the last axiom α is added. This conclusion will be conflict with the fact that the previous one doesn't entail η. Therefore, the last axiom α appended to the final $\mathcal{O}'$ by Function 1 must belong to any justification w.r.t. η and the final $\mathcal{O}'$, and it must be critical according to Definition 2. □

Take the ontology in Fig. 1 as an example, in which we try to detect a critical axiom w.r.t a sub-ontology of $\mathcal{O}_1$ and η_1. According to Function 1, firstly an axiom is selected (e.g., α_1 is selected) from $\mathcal{O}_1$ into $\mathcal{O}'$. η_1 is not entailed by $\mathcal{O}'$ at this moment. Then we keep on adding α_2 into $\mathcal{O}'$, and the entailment relationship will be hold between η_1 and updated $\mathcal{O}'$. Thus the procedure stops, and the final $\mathcal{O}'$ includes axioms α_1 and α_2. α_2, the critical axiom w.r.t. $\mathcal{O}'$ and η_1, is returned by Function 1. Notice that α_2 is a **CRIT**(η_1, $\mathcal{O}'$). Similarly, α_1 will be the critical axiom if α_2 is selected at first, and α_1 is added later during this process. In this case, α_1 is a **CRIT**(η_1, $\mathcal{O}'$).

3.3 Construction of a justification by recursively detecting critical axioms

In fact, a justification is a special minimal ontology which entails axiom η. When we consider a justification as the whole ontology, every axiom in this ontology also belongs to the sole justification at the same time. Proportion 2 is given as below.

Proposition 2 $\forall\alpha \in \mathcal{O}$ is critical w.r.t. $\mathcal{O}$ and η **iff** $\mathcal{O}$ is the justification w.r.t. $\mathcal{O}$ and η.

Proof First, the proposition is necessary. Given an ontology $\mathcal{O}$ itself is the justification w.r.t. $\mathcal{O}$ and η. At this moment, there must be only one justification for $\mathcal{O}$ and η. In other words, **ALL_JUST**($\eta, \mathcal{O}$)={{$\mathcal{O}$}}. Every axiom belonging to $\mathcal{O}$ also belongs to this justification w.r.t. $\mathcal{O}$ and η. Therefore every axiom belonging to $\mathcal{O}$ must be critical w.r.t. $\mathcal{O}$ and η according to Definition 2.

Conversely, the proposition is sufficient. Assuming that there is another ontology $\mathcal{O}' \subset \mathcal{O}$ which is also the justification w.r.t. $\mathcal{O}$ and η when $\forall\alpha \in \mathcal{O}$ is critical w.r.t. $\mathcal{O}$ and η. According to Definition 1, $\mathcal{O}$ will not be the justification because $\mathcal{O}$ is not the smallest anymore. At this moment, axioms which belong to $\mathcal{O}$ but not belong to $\mathcal{O}'$ are not critical w.r.t. $\mathcal{O}$ and η according to Definition 2. This is conflict with the precondition of the proposition. So no such sub-ontology $\mathcal{O}'$ exists. $\mathcal{O}$ must be the only justification w.r.t. $\mathcal{O}$ and η when $\forall\alpha \in \mathcal{O}$ is critical w.r.t. $\mathcal{O}$ and η. □

According to Proposition 2, we can devise a recursive procedure to construct all elements of some justification as shown in Algorithm 1 which recalled Function 1. $\mathcal{O}$ and $\mathcal{O}'$ are updated by $\mathcal{O}'$ and $\mathcal{J}$ (a set of critical axioms w.r.t. η and $\mathcal{O}'$) each time before recalling Function 1, respectively, in order to pinpoint the critical axiom in a more precise and smaller scale. In the loop program, each round will find a new critical axiom for η_1 and current ontology $\mathcal{O}$. At the end of each round, $\mathcal{O}$ will become smaller and $\mathcal{J}$ will be bigger than that from the previous round. Finally, they will equal to the same set of axioms (i.e., justification) when the whole program is over.

Algorithm 1 Construct_Justification

Input: Ontology $\mathcal{O}$, Entailment Axiom η;
Output: Justification $\mathcal{J}$;
1: $\mathcal{J} \leftarrow \emptyset$
2: $\mathcal{O}' \leftarrow \emptyset$
3: **while** $|\mathcal{J}| < |\mathcal{O}|$ **do**
4: Find_Critical_Axiom($\mathcal{O}$,$\mathcal{O}'$,η)
5: $\mathcal{J} \leftarrow \mathcal{J} \cup \{\alpha\}$
6: $\mathcal{O} \leftarrow \mathcal{O}'$
7: $\mathcal{O}' \leftarrow \mathcal{J}$
8: **end while**
9: **return** $\mathcal{J}$

Theorem 1 $\mathcal{J}$ returned by Algorithm 1 is a justification w.r.t. ontology $\mathcal{O}$ and axiom η.

Proof Both justification $\mathcal{J}$ and critical axiom set $\mathcal{O}'$ are initialized to be empty in steps 1 and 2. When the algorithm runs into the recursive body, it will recall Function 1 in step 4 of this algorithm. The last axiom α appended by Find_Critical_Axiom ($\mathcal{O}$, $\mathcal{O}'$, η) must be a critical axiom for $\mathcal{O}$ according to Proposition 1. So it has to be the element of some justification and expanded to $\mathcal{J}$ in step 5. Then $\mathcal{O}$ is replaced by $\mathcal{O}'$, and $\mathcal{O}'$ is replaced by $\mathcal{J}$ sequently in steps 6 and 7. It is guaranteed to find a new element of a justification

in a smaller ontology during the next recursion.

For the circular body described from steps 3 to 8 in Algorithm 1, a new element (i.e., a critical axiom) of justification $\mathcal{J}$ will be explored in each round. If η is entailed by $\mathcal{J}$ at the start of a round, $\mathcal{O}$ will still be equal to $\mathcal{J}$ at the end of the same round. The recursion will be ended when $\mathcal{O}$ equals to $\mathcal{J}$ according to the critical condition of step 3. At this moment, every axiom is critical w.r.t. $\mathcal{O}$ and η. According to Proposition 2, $\mathcal{J}$ will be the justification w.r.t. η and current updated $\mathcal{O}$. It also be the justification for η and original $\mathcal{O}$. □

An illustration for the process of Algorithm 1 using the example from Fig. 1 is as the following. The input is ontology $\mathcal{O}_1 = \{\alpha_1, \alpha_2, \alpha_3, \alpha_4\}$ as $\mathcal{O}$ and axiom $\eta_1 : A \sqsubseteq C$ as η, and the output is a $\mathbf{JUST}(\mathcal{O}_1, \eta_1) = \{\alpha_1, \alpha_2\}$.

Table 1 illustrates the ending status of each round for the recursive execution (from steps 3 to 8 in Algorithm 1). All rounds are recorded in the first column of Table 1, such as Round 1 and Round 2. The item "α" appeared in the third column is the critical axiom calculated by Find_Critical_Axiom($\mathcal{O}, \mathcal{O}', \eta$) (i.e., step 4). At the end of Round 1, α_2 is critical returned by Function 1 for η_1 in $\mathcal{O}'$ including α_1 and α_2. Then α is added into the justification $\mathcal{J}$ (step 5). In order to select a new axiom from $\mathcal{O}\backslash\mathcal{O}'$ for detecting the next critical axiom, we would update $\mathcal{O}$ and $\mathcal{O}'$ (steps 6 and 7). $\mathcal{O}$ and $\mathcal{O}'$ are updated as $\{\alpha_1, \alpha_2\}$ and $\{\alpha_2\}$, respectively, at the end of Round 1. In a similar way, we can obtain the status of Round 2. At the end of Round 2, $\mathcal{J}$ equals to $\mathcal{O}$, and the recursion stops. $\mathbf{JUST}(\mathcal{O}_1, \eta_1) = \{\alpha_1, \alpha_2\}$ which is assigned in the lower right corner of Table 1 is returned by this program.

Table 1 Illustration of algorithm 1 using example from Fig. 1

	$\mathcal{O}$	$\mathcal{O}'$	α	$\mathcal{J}$
Initialization	$\alpha_1, \alpha_2, \alpha_3, \alpha_4$	$\emptyset$	$\emptyset$	$\emptyset$
Round1	α_1, α_2	α_2	α_2	α_2
Round2	α_1, α_2	α_1, α_2	α_1	α_1, α_2

4 Optimization of the recursive constructing procedure

As mentioned in the motivation section, the naive strategy given in [2] and [3] for an ontology w.r.t. unsatisfiable concept C including m axioms only requires m times of satisfiability decision to detect a certain MUPS. We interchange the concept of MUPS and justification since they could be transferred equivalently. Compared to such naive shrink strategy, times of satisfiability decision about Algorithm 1 is $m(m+1)/2$ in the worst case.

In fact, the worst case rarely occurs in practice. On one hand it is not necessary to traverse every axiom of the unsatisfiable ontology in one round (mentioned in Section 4.1). On the other hand, it can be transferred into an equivalent incremental reasoning problem instead of performing a large number of independent satisfiability tests in one round (mentioned in Section 4.2). This can be tackled in an easier and more efficient manner.

4.1 Axiom selection function

Our algorithm described above is greedy in some sense. Axioms in the ontology will be added all the time to a set of axioms unless such set is proved to be unsatisfiable. During this procedure, axiom selection has a great impact on performance in terms of speed. Some simple optional automatic axiom sorting methods are easy to be implemented, such as base on weight (i.e., sorting axioms selected by the sum of number of occurrences in the ontology of all concepts of the axiom), base on length (i.e., sorting axioms with the shortest ones first), and so on. In contrast to these common indicators, we here adopt a special strategy named "selection function" for OWL ontology which is proposed by Huang [21]. In the reasoning of inconsistency, [21] used syntactic relevance to define a selection function to extend the query "$\Sigma \models \phi$?" until the reasoning process can obtain an answer to the query ϕ. In our work, we adopt it to select and determine the order of axioms expanded. The selection function defined in the following will stop until the ontology becomes unsatisfiable.

Definition 3 (Selection function [21]) $s(\Sigma, \phi, k) = \{\psi \in \Sigma \mid \psi$ is directly relevant to $s(\Sigma, \phi, k-1)\}$, for $k > 1$.

In Definition 3, s is the name of selection function, Σ is an ontology, both ψ and ϕ are axioms of ontology. Two axioms (or ontologies) are directly relevant if and only if there are common individuals, concepts or roles. Selection function will run k times until we obtain enough axioms for satisfiability checking.

4.2 Incremental reasoning for satisfiability decision

In the procedure for critical axiom detection, besides selecting proper axioms, the reasoning mission about satisfiability decision for a series of continually updated ontologies can be treated more efficiently. In order to transform it into an incremental reasoning problem, such series of continually updated ontologies are defined formally as the following:

Definition 4 (Ontologies chain) For an ontology $\mathcal{O}$, S

which is a subset of $2^{\mathcal{O}}$ is called an ontologies chain, if an axioms inclusion $\subset_{axiom}$ on S is a quasi and totally ordering.

Definition 5 (Incremental debugging ontologies, IDOs) An ontologies chain is called an incremental debugging ontologies when the axiom η is only entailed by its greatest element. Each element of IDOs is called debugging ontology (DO)

According to Definition 5, an IDOs can be ordered as a sequence $< \mathcal{O}_1, \mathcal{O}_2, \ldots >$ by Hasse Diagrams. Each element of this sequence does not entail η except the last one. Actually, if only one axiom from current ontology is added to the next one each time, the critical axiom can be detected by the last two elements of the sequence. We consider to build a growth procedure by adding only one axiom (step size). The added axiom is called **incremental axiom**, noted as α_i when it's added to $\mathcal{O}_i$. In this procedure, axiom η should not be entailed by any $\mathcal{O}_i$, but entailed by the last one from the corresponding IDOs. The last element of corresponding sequence of added axioms sequence $< \alpha_1, \alpha_2, \ldots >$ is a critical axiom.

Proposition 3 The last axiom appended to the incremental debugging ontologies is critical if axiom η is not entailed by any $\mathcal{O}_i$ except the last one.

This incremental reasoning problem is possible to be tackled by using cached results from previous model w.r.t. the former ontology to demonstrate the satisfiability of the following ontology. The similar optimization technique has been adopted in [22–24] to detect "obvious" satisfiability in order to resolve classification problem. In our problem, the situation is different in two main aspects. First of all, we decide the satisfiability about the whole knowledge base, not only about concepts. Second, there is only one model known in the IDOs, not two in the classification problem. Therefore, we cannot use caching optimization technique directly.

In addition, attention should be paid to the implementation of entailment reasoning. Normally, the entailment is equivalently transferred into unsatisfiability decision for a concept w.r.t. an ontology firstly (i.e., finding MUPS problem) [2]. Furthermore, it is equivalently transferred into satisfiability decision w.r.t. some concept by tableaux expansion [25].

To solve this problem, we firstly obtain the final ABox of the current DO according to its model. Such ABox is also called "pseudo model" in [26]. Then we transform the entailment decision for the next DO into the satisfiability of ABox for the current DO w.r.t. the increment axiom α_i. We call it incremental reasoning about IDOs. It uses cheaper reasoning cost (does not need to expand the whole knowledge by tableaux rules again) instead of stand entailment tests. The incremental reasoning procedure for detecting a critical axiom is described in the following Function 2.

Function 2 InRe_Critical_Axiom($\mathcal{O}$,$\mathcal{O}'$,η)

Input: Ontology $\mathcal{O}$, $\mathcal{O}' \subseteq \mathcal{O}$ is a DO, Entailment Axiom η;
Output: **CRIT**(η, $\mathcal{O}'$) and $\mathcal{O}'$;

1: $\mathcal{FABOX} \leftarrow$ getFinalAbox($\mathcal{O}' \cup \mathcal{G}(\eta)$)
2: **while** η is not entailed by $\mathcal{O}'$ **do**
3: select some axiom α from $\mathcal{O}\backslash\mathcal{O}'$ according to the select function
4: **if** $\mathcal{FABOX}$ is satisfiable w.r.t. $\{\alpha\}$ **then**
5: $\mathcal{FABOX} \leftarrow$ getFinalAbox($\mathcal{O}' \cup \{\alpha\}$)
6: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{\alpha\}$
7: **else**
8: **if** η isn't entailed by $\mathcal{O}' \cup \{\alpha\}$ **then**
9: $\mathcal{FABOX} \leftarrow$ *getFinalABox*($\mathcal{O}' \cup \{\alpha\}$)
10: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{\alpha\}$
11: **end if**
12: **end if**
13: **end while**
14: $\mathcal{O}' \leftarrow \mathcal{O}' \cup \{\alpha\}$
15: **return** *last appended*(α) and $\mathcal{O}'$

In Function 2, target axiom η is transformed into the corresponding concept by the function $\mathcal{G}$ in step 1. $\mathcal{G}$ has been defined in [26] in order to transfer entailment tests into the satisfiability decision problem. For example, a final ABox about knowledge base $\mathcal{O}' \cup \{\alpha\}$ will be obtained by tableaux expansion when $\mathcal{O}' \cup \{\alpha\}$ is satisfiable. This calculus procedure is denoted as a function *getFinalABox*( ). Then we select an added axiom as the incremental axiom by a selection function which is mentioned in Section 4.1. In step 4, we keep on deciding the satisfiability of the final ABox about the current DO w.r.t. the incremental axiom a instead of entailment test between $\mathcal{O}' \cup \{\alpha\}$ and η. Updating corresponding final ABox and DO in steps 5 and 6 if satisfiability attains. Otherwise, we are uable to announce that $\mathcal{O}' \cup \{\alpha\}$ is unsatisfiable. Because there may exist a model constructed by tableaux expansion rules for the initial knowledge $\mathcal{O}' \cup \{\alpha\}$ but not for the final ABox w.r.t. α. In other words, it is sound but we cannot prove that is also completed. So we should test the entailment in step 8, and execute corresponding operations in steps 9 and 10. Recurring such procedure until we find the critical axiom α and the corresponding ontology $\mathcal{O}'$ (steps 14 and 15).

If Function 2 replaces Function 1 which is recalled in step 4 for Algorithm 1, an optimized procedure for detecting α justification will be obtained.

5 Experiments

In the previous sections, we have elaborated on the theory,

improvements, designment, and optimizations for detecting a justification for OWL ontologies by critical axioms. In our implementation, a solver is utilized to detect justifications of some public ontologies and their target axioms. The experimental results demonstrate the effectiveness of our proposed method. The properties of ontologies and testing environment are described in Section 5.1. Experiments for a single justification and all justifications are conducted and corresponding results are reported in Section 5.2 and Section 5.3, respectively.

5.1 Ontologies and testing environment

In order to evaluate our method efficiently, we select some real-world ontologies. Some come from WebProtégé repositories, and others come from similar testing works such as [2, 25, 27]. Details for different tested ontologies are given in Table 2. The first column records the name of ontologies. The second one describes all kinds of characteristics in terms of the description logic (DL) used to formulate the ontology. The third column C/P/I summarizes information about the number of concepts, properties, and individuals contained by the corresponding ontology. The last column N gives the number of entailment problems, and such number corresponds to the number of unsatisfiable classes which are special entailments for ontologies debugging tasks. Most ontologies such as MadCow, dbPediea-2014, Tambis, and so on have unsatisfiable classes. While the current version of SWEET Ontology and University Ontology does not exist unsatisfiable concepts. So we choose some concepts in these two ontologies to generate entailments randomly as testing tasks. The counts of entailments about SWEET and University Ontology are 30 and 10, respectively.

Table 2 Description of tested ontologies

Ontology	DL	A	C/P/I	N
1.MadCow	$\mathcal{ALCHOIN}^{(D)}$	109	53/29/13	1
2.dbPedia-2014	$\mathcal{ALCHF}^{(D)}$	6,772	828/3035/1	2
3.SWEET-JPL	$\mathcal{ALCHOF}^{(D)}$	3,833	1537/121/50	30
4.Tambis	$\mathcal{SHIN}^{(D)}$	795	392/112/0	144
5.University	$\mathcal{SHI}^{(D)}$	117	43/44/1555	10
6.Economy	$\mathcal{ALCH}^{(D)}$	663	338/65/482	51
7.Transportation	$\mathcal{ALCN}^{(D)}$	1,282	444/105/183	62

We implement algorithms and related optimizations described in this paper in Java code base on OWL API. All evaluations are carried out on a laptop with 2.3 GHz Intel(R) Core(TM)2 Duo CPU and 4.0 GB of RAM using 32 bit operating system Windows 7.

5.2 Experiment for a single justification

First of all, we compare performance between the basic expansion algorithm and traditional prune algorithm. Expansion and prune procedures only select one axiom to deal with each time and only test usual satisfiability (not increment reasoning satisfiability mentioned in Section 4.2.). Here we adopt the select function described in Section 4.1 for the basic expansion method to compare its performance with the traditional prune method.

Experiments results for the basic expansion method and prune method are summarized in Tables 3–4. We test entailment tasks for every ontology listed in Table 2. The running time and times of satisfiability decision to be called are filled separately in each row. The unit of running time is “ms”, and the unit of calling times of satisfiability decision is “times”. Results in Table 3 show that the basic expansion method is generally worse than traditional prune method in terms of time spent and calling times. The reason is that there are multiple rounds tests in our expansion method. The count of rounds for the expansion method equals to the number of axioms included in a justification, while pruning method only needs one round to shrink. Therefore, our basic expansion method spends more time and calling times, even if our method can avoid testing every axiom in one round. However, our method with increment reasoning optimization stated in Section 4.2 is better than the basic pruning method. We can confirm this conclusion by comparing the 4th line in Table 3 to the 2nd line in Table 4.

Table 3 Evaluation between basic expansion and prune

(units: ms/times)

Ontology ID	1	2	3	4	5	6	7
Expansion method	59	347	3,843	77,014	1,694	1,341	6,682
	29	276	2,217	36,978	937	1,145	5,719
Prune method	45	178	2,954	57,824	1,370	882	4,998
	10	121	1,654	21,547	618	765	3,998

In fact, modern ontology reasoners, such as Pellet or Fact++, does not adopt the traditional prune strategy anymore. Most of them use the sliding windows strategy to perform a fast pruning. In such prune procedure, multiple axioms (not only one axiom) are selected at the same time for one testing satisfiability. Thus times of calling satisfiability decision will be reduced greatly. Does our expansion method still have some advantage to compare with prune method under such situation? The good news is our expansion method can be optimized by using increment reasoning described in Section 4.2 (fast pruning or slow pruning cannot adopt).

The speed of increment reasoning is obviously faster than the usual satisfiability decision, so our expansion method is likely to have superior performance. In order to confirm it, we conduct experiment to compare the expansion method with increment reasoning optimization and the fast prune method with sliding windows optimization. The results are displayed in Table 4.

Table 4 Incremental reasoning vs. Sliding windows optimization

(units: ms/times)

Ontology ID	1	2	3	4	5	6	7
Expansion with INR	37	58	862	3,564	481	387	1,194
	8	32	639	2,184	173	402	579
Prune with WIN	43	87	1,124	4,811	609	512	2,010
	10	51	731	2,765	201	564	1,874

The test terms in Table 4 are similar with that in Table 3. We also test the running time and times of satisfiability decision to be called during implementation processes. The times of calling satisfiability test which is executed by expansion method with increment reasoning optimization only are recorded by normal test. In other words, we ignore to count the times of increment reasoning. Because the spending of such special reasoning is small enough compared to common satisfiability test of the whole knowledge base. Obviously, the efficiency of pruning method with sliding windows shown in Table 4 is better than the slow pruning method in Table 3. Dealing multiple axioms instead of a single axiom each time has great impact on the efficiency. However, it does not has advantage when comparing to our expansion method with increment reasoning optimization. The results in Table 4 show that the reasoning time spent is also an important element for the improvement of efficiency.

5.3 Experiment for all justifications

Given an initial justification, we can use a variation of the classical Hitting Set Tree (HST) algorithm [27] to compute the remaining ones. To begin with, we find any single justification and set it as the root node of the HST. Then each of axioms in the justification is removed individually, thus new branches of the HST are created, and we find new justifications along these branches on the fly (using our proposed algorithm in this paper) in the modified ontology. This process needs to be exhaustively done in order to compute all justifications. The details about finding all justifications are stated in [2]. In this section, we use such a variation of the HST algorithm to extract all justifications for various ontologies mentioned in Table 2. The statistics of attributes of justifications for various ontologies are shown in Table 5.

Table 5 Justifications statistics about ontologies

Ontologies	No. of JUSTs		Size of JUST	
	mean	max	mean	max
1	1	1	4	4
2	1	1	4	4
3	1.63	3	3.96	5
4	–	–	–	–
5	1.5	2	4.25	6
6	1.29	2	3.74	6
7	2.18	5	5.58	9

The first row of "No. of JUSTs" is the mean number of justifications for each target axiom, and the second row presents the target axiom which owns the most justifications. The first row of "Size of JUST" is the mean number of axioms for each justification, and the second row presents the justification which owns the most axioms. According to Table 5, the numbers of justifications and axioms of MadCow and dbPedia-2014 are both less than other ontologies.

The time spent with the extraction of every ontology appeared in Table 2 is plotted in Fig. 3. Deep grey lines represent the running time by Pellet which are mature tools to find all justifications for OWL ontologies. Shallow grey lines represent the running time by a variation of HST algorithm invoking our proposed method in this paper. Among 7 ontologies, notice that for dbPedia-2014, even if there are only two target axioms for its testing, it is still time-consuming since the justifications of this ontology are more complicated than others. In addition, Tambis ontology owns 144 target axioms. The testing has exhausted the computer memory, thus does not produce results as shown in line 4 of Table 5. The same story happens in Fig. 3, where the time spent for Tambis ontology is ignored. As a result, we observe that invoking our method to compute all justifications achieves a better performance than the traditional methods, i.e., Pellet.

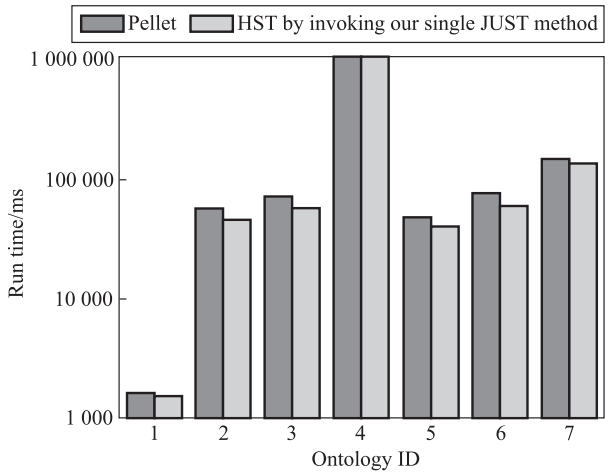

Fig. 3 Run time for Pellet vs. HST based on our single JUST method

6 Conclusion

In this paper, a novel approach is proposed to detect one justification for OWL ontologies. Compared to the traditional pruning strategy for reducing axioms, our proposed method expands axioms directly. This is realized by detecting critical axiom through a series of definitions, properties, theorems, and algorithms. At the same time, some optimization techniques such as select function and increment reasoning have been adopted to improve this approach in further. The experiment results show that our method wins in efficiency.

In our experiments, we simply invoke our method via using Hitting-tree algorithm to detect all justifications. That is to say, we only substitute the traditional pruning method with the proposed method in combining with Hitting-tree algorithm, which does not fully display advantages and performance of our proposed method. We believe there are a lot of possible improvement which is worthy to be explored in our future work.

Acknowledgements Research presented in this paper was partially supported by the National Natural Science Foundation of China (Grant Nos. 61672261, 61502199). It's also funded by China Scholarship Council (201506175028) for the first author of this paper. We would like to be grateful to the partners in the laboratory who have given our generous support and helpful advice for this study. Specially, thanks are due to Assistant Professor Jiafeng Xie for assistance with the experiments and proofreading the manuscript.

References

1. Schlobach S, Cornet R. Non-standard reasoning services for the debugging of description logic terminologies. In: Proceedings of the 18th International Joint Conference on Artificial Intelligence. 2003, 355–362
2. Kalyanpur A, Parsia B, Horridge M, Sirin E. Finding all justifications of OWL DL entailments. In: Proceedings of the 6th International the Semantic Web and 2nd Asian Conference on Asian Semantic Web Conference. 2007, 267–280
3. Schlobach S, Huang Z S, Cornet R, Harmelen F V. Debugging incoherent terminologies. Journal of Automated Reasoning, 2007, 39(3): 317–349.
4. Baader F, Penaloza R. Automata-based axiom pinpointing. Journal of Automated Reasoning, 2010, 45(2): 91–129
5. Ma Y, Penaloza R. Towards parallel repair: an ontology decomposition-based approach. In: Proceedings of the 27th International Workshop on Description Logics. 2014, 633–645
6. Teymourlouie M, Zaeri A, Nematbakhsh M A, Staab S. Detecting hidden errors in an ontology using contextual knowledge. Expert Systems with Applications, 2018, 95: 312–323
7. Xue H, Qin B, Liu T. Topic hierarchy construction from heterogeneous evidence. Frontiers of Computer Science, 2016, 10(1): 136–146
8. Wu L, Su K, Han Y. Reasoning about knowledge, belief and certainty in hierarchical multi-agent systems. Frontiers of Computer Science, 2017, 11(3): 499–510
9. Ouyang D, Cui X, Ye Y. Integrity constraints in OWL ontologies based on grounded circumscription. Frontiers of Computer Science, 2013, 7(6): 812–821
10. Zhang Y, Ouyang D, Ye Y. Glass box debugging algorithm based on unsatisfiable dependent paths. IEEE Access, 2017, 5: 18725–18736
11. Zhang Y, Ouyang D, Ye Y. An optimization strategy for debugging incoherent terminologies in dynamic environments. IEEE Access, 2017, 5: 24284–24300
12. Friedrich G, Shchekotykhin K M. A general diagnosis method for ontologies. In: Proceedings of the 4th International Conference on the Semantic Web. 2005, 232–246
13. Schlobach S. Diagnosing terminologies. In: Proceedings of the 20th National Conference on Artificial Intelligence. 2005, 670–675
14. Shchekotykhin K M, Friedrich G, Fleiss P, Rodler P. Interactive ontology debugging: two query strategies for efficient fault localization. Journal of Web Semantics, 2012, 12: 88–103
15. Jannach D, Schmitz T, Shchekotykhin K M. Parallel model-based diagnosis on multi-core computers. Journal of Artificial Intelligence Research, 2016, 55: 835–887
16. Horridge M, Bauer J, Parsia B, Sattler U. Understanding entailments in OWL. In: Proceedings of the 5th OWLED Workshop on OWL. 2008, 26–27
17. Bail S, Parsia B, Sattler U. Declutter your justifications: determining similarity between OWL explanations. In: Proceedings of the 1st International Workshop on Debugging Ontologies and Ontology Mappings. 2012, 13–24
18. Reiter R. A theory of diagnosis from first principles. Artificial Intelligence, 1987, 32: 57–95
19. Maaren H V, Wieringa S. Finding guaranteed MUSes fast. In: Proceedings of the 11th International Conferences on Theory and Applications of Satisfiability Testing. 2008, 291–304
20. Kullmann O, Lynce I, Marques J. Categorisation of clauses in conjunctive normal forms: minimally unsatisfiable sub-clause-sets and the lean kernel. In: Proceedings of the 9th International Conference of Theory and Applications of Satisfiability Testing. 2006, 22–35
21. Huang Z S, Harmelen F V, Teije A Y. Reasoning with inconsistent ontologies: framework, prototype, and experiment. In: Davies J, Studer R, Warren P, eds. Semantic Web Technologies: Trends and Research in Ontology-based Systems. Hoboken: John Wiley & Sons, Inc., 2006
22. Horrocks I. Implementation and optimisation techniques. In: Baader F, Calvanese D, McGuinness D, Nardi D, Schneider P F, eds. The Description Logic Handbook: Theory, Implementation, and Applications. 2nd ed. London: Cambridge University Press, 2007
23. Parsia B, Halaschek C, Sirin E. Towards incremental reasoning through updates in OWL-DL. In: Proceedings of the 15th International Conference of World Wide Web. 2006
24. Grau B C, Halaschek C, Kazakov Y, Suntisrivaraporn B. Incremental classification of description logics ontologies. Journal of Automated Reasoning, 2010, 44(4): 337–369
25. Horrocks I, Schneider P F. Reducing OWL entailment to description logic satisfiability. In: Proceedings of the 2nd International Conference on the Semantic Web. 2003, 17–29

26. Haarslev V, Moller R, Turhan A Y. Exploiting pseudo models for TBox and ABox reasoning in expressive description logics. In: Proceedings of International Joint Conference on Automated Reasoning. 2001, 61–75

27. Ji Q, Gao Z, Huang Z, Zhu M. Measuring effectiveness of ontology debugging systems. Knowledge-Based Systems, 2014, 71: 169–186

Yuxin Ye received his PhD degree in computer software and theory from Jilin University, China in 2010. He is currently an associate professor in the College of Computer Science and Technology, Jilin University, China. He also serves on Key Laboratory of Symbolic Computation and Knowledge Engineering (Jilin University), Ministry of Education, China. He has more than 10 years of experience in Ontology Engineering and Semantic Web research and has more than 40 publications in these areas. He is a Member of China Computer Federation (CCF). His main research interests include semantic Web, ontology engineering, and knowledge graph.

Xianji Cui received her PhD degree in computer software and theory from Jilin University, China in 2014. She is currently a lecture in College of Information and Communication Engineering, Dalian Minzu University, China. Her main research interests include semantic Web and ontology engineering.

Dantong Ouyang received her PhD degree in computer software and theory from Jilin University, China in 1998. She is currently a professor and PhD supervisor in the College of Computer Science and Technology, Jilin University, China. She is a senior member of China Computer Federation (CCF). She also serves on some academic organizations, such as CCF TCAIPR, CCF TTCS, CAAI KE&DS, and so on. Her main research interests include artificial intelligence, automatic reasoning, model based diagnosis, constraint problem, and so on.