

# MUPS identification based on discrimination rules

Wang Wenbin Yang Fan Rao Yimei Chen Qiushuang

(College of Information Technical Sciences, Nankai University, Tianjin 300071, China)

**Abstract:** The minimal unsatisfiability-preserving sub-TBoxes (MUPS) of an unsatisfiable class  $C$  identified by two equivalent transformations, axiom splitting and ontology reduction, and three discrimination rules comprise minimal sets of axioms which support the unsatisfiability. Discrimination rules classify all MUPS into three types based on the transitivity of unsatisfiability, fully dependent on  $C$  (MUPSf), transitively dependent on  $C$  (MUPSt) and uncertainly dependent on  $C$  (MUPSu). The results show that the number of MUPSt is frequently a large fraction of the total number of all MUPS, but only MUPSf catches the root error of  $C$ . Modelers and domain experts conduct iterative repair work effectively, considering only MUPSf in each iteration. The classification shows the significance for the evaluation of the quality of ontologies from the perspective of maintenance and for repair work.

**Key words:** ontology debugging; minimal unsatisfiability-preserving sub-TBoxes (MUPS); discrimination rules

Ontologies play a crucial role in the semantic web and evolve over time. One of the major problems of evolving ontologies is the potential introduction of inconsistencies. As the ontology grows larger and more complex, it may no longer be correct or consistent. When dealing with inconsistent ontologies with unsatisfiable classes, it is crucial to know what causes the inconsistencies before repairing them.

MUPS<sup>[1]</sup> is a minimal set of axioms which can support an unsatisfiable class. The prerequisite for fixing an unsatisfiable class is to find the MUPS of the class, i. e., the axioms causing the class to be unsatisfiable. There are three kinds of approaches to finding MUPS: black-box approach, glass-box approach and a mix of them. While the glass-box approach is based on modifying the internals of the description logics (DLs) reasoner, the black-box approach uses the DLs reasoner solely as a subroutine and does not need to modify the reasoner itself.

The black-box approach is reasoner-independent. The debugger does not need to know any detail of the internal tableau algorithm or the optimization of the reasoner. A simple expand-shrink strategy was proposed to find a MUPS of an unsatisfiable class in Ref. [2]. In the expanding phase, the approach expands an empty set using axioms from the original knowledge base (KB) until the class is unsatisfiable in the KB. Then in the shrinking phase, this set is pruned to achieve a minimal set of axioms responsible for the unsatisfiability. A bottom-up approach<sup>[3]</sup> improved on the expand-shrink strategy<sup>[2]</sup>, which uses a relevance-based selection

function to find a set of inconsistent axioms. Although both the above methods can derive correct results, they are time consuming. A heuristic was put forward in Ref. [4], which includes two alternate expand-shrink steps. Based on a pre-defined set of rules to detect commonly occurring error patterns, the heuristic can obviously reduce the calculation cost. However, it is incomplete.

The glass-box approach is built on existing tableau-based decision procedures for expressive DLs; it is reasoner-dependent. The implementation of the glass-box approach requires a non-trivial modification of the internals of the reasoner. Complete algorithms for unfoldable ALC-TBoxes were developed using Boolean methods<sup>[1,5]</sup>. As an extension work, Kalpanpar<sup>[6]</sup> extended the work in Ref. [1] to the more expressive logic SHIF<sup>[7]</sup>, which corresponds to OWL-Lite. Lee et al.<sup>[8]</sup> also extended the work in Ref. [1] to make the general TBoxes capable of being handled.

A hybrid approach presented in Ref. [9] is based on the tableau decision procedure for class satisfiability in SHION. The approach uses the tableau algorithm in place of the first step in Ref. [2] to obtain a much smaller set of axioms than the original KB, bringing the runtime of the expanding process be obviously reduced.

All the above approaches cannot ensure obtaining all the MUPS of an unsatisfiable class. It is very possible that the class is still unsatisfiable after the axioms of one MUPS are repaired, due to the clash among the axioms of another MUPS. Reiter's hitting set tree (HST)<sup>[10]</sup> can derive all the MUPS of an unsatisfiable class. However, the HST needs a MUPS as an input, because it finds all minimal hitting sets based on a given collection of conflict sets.

In this paper, a MUPS identification approach based on discrimination rules (MIDR) for finding all MUPS is developed, which is totally reasoner independent. MIDR is unlike HST in that it needs a MUPS as an input. Most importantly, MIDR can classify all MUPS into three types, which is very important for evaluating the quality of an ontology and for the repair work. A prototype system OntoMD has been implemented based on which the experiments are performed.

## 1 Preliminaries

### 1.1 Description logics and OWL

In DLs, concepts are interpreted as subsets of a domain, and roles as binary relations. A KB comprises two components, TBox and ABox. TBox introduces the terminology, i. e., the vocabulary of an application domain, while ABox contains assertions about named individuals in terms of this vocabulary.

The web ontology language (OWL) is a world wide web consortium standard for writing ontologies or formal vocabularies by which computers can understand the data with semantic annotation. From a semantic point of view, there is a

Received 2008-04-15.

**Biographies:** Wang Wenbin (1982—), male, graduate; Chen Qiushuang (corresponding author), female, doctor, professor, chenqs@nankai.edu.cn.

**Foundation item:** The Science and Technology Development Program of Tianjin (No. 06YFGZGX05900).

**Citation:** Wang Wenbin, Yang Fan, Rao Yimei, et al. MUPS identification based on discrimination rules[J]. Journal of Southeast University (English Edition), 2008, 24(3): 372 – 376.

strong relationship between DLs and OWL<sup>[11]</sup>.

An OWL ontology consists of classes, properties and individuals. In the semantic web, ontologies provide shared data structures to exchange information between agents and can be used for knowledge-based services.

## 1.2 Unsatisfiable class

An unsatisfiable class is one that cannot possibly have any individuals; it is equivalent to the empty set (in OWL terms, to the bottom class, OWL: nothing).

Consider three types of class axioms: subclass axiom, equivalent class axiom and disjoint class axiom:

$$A \subseteq B, A \equiv B, A \subseteq \neg B \quad (A \text{ and } B \text{ refer to OWL classes})$$

If the class  $B$  is unsatisfiable, the subclass/the equivalent class  $A$  is also unsatisfiable, with the exception of the disjoint class axiom which is uncertain. Moreover, any existential restriction and hasValue restriction (unnamed class) with an unsatisfiable filler are themselves unsatisfiable. Any restriction with an unsatisfiable domain or range is unsatisfiable, excepting a universal restriction. Any subclass of an unsatisfiable class is unsatisfiable<sup>[4]</sup>.

## 1.3 MUPS

A MUPS of an unsatisfiable concept (class)  $C$  w. r. t. a knowledge base (or in other words, an OWL ontology)  $K$  can be defined as follows<sup>[11]</sup>:

**Definition 1** (MUPS) Let  $C$  be a concept, which is unsatisfiable w. r. t. a knowledge base  $K$ .  $K' \subseteq K$  is a MUPS of  $C$  in  $K$  if  $C$  is unsatisfiable in  $K'$ , and  $C$  is satisfiable in every  $K'' \subset K'$ .

The following example illustrates the definition of MUPS:

$K_1$  is a knowledge base;  $A$ ,  $B$  and  $C$  are atomic concepts; and  $A_1, \dots, A_5$  define concept names:

$$ax_1: A_1 \subseteq \neg A \cap A_2 \cap A_3$$

$$ax_2: A_2 \subseteq A \cap A_4$$

$$ax_3: A_3 \subseteq A_4 \cap A_5$$

$$ax_4: A_4 \subseteq \forall s. B \cap C$$

$$ax_5: A_5 \subseteq \exists s. \neg B \cap C$$

By a DL reasoner such as Pellet, the set of unsatisfiable concepts can be obtained as  $\{A_1, A_3\}$ . The MUPS of the unsatisfiable concepts is

$$\text{MUPS}(A_1, K_1) = \{\{ax_1, ax_2\}, \{ax_1, ax_3, ax_4, ax_5\}\}$$

$$\text{MUPS}(A_3, K_1) = \{\{ax_3, ax_4, ax_5\}\}$$

In the remainder of this paper,  $\text{MUPS}(C, K)$  denotes the set of all MUPS for unsatisfiable concepts  $C$  in  $K$ .

## 1.4 Classification of MUPS

Identifying all MUPS of an unsatisfiable class is a non-trivial work, because by considering the interaction among the axioms of the MUPS, modelers will know why the class is unsatisfiable. In our view, these MUPS can be classified into three types, each MUPS type makes a different contribution to the repair work.

to the repair work.

**Definition 2** Let  $C_1, C_2, \dots, C_n$  be a set of unsatisfiable classes w. r. t. an ontology  $O$ ,  $K'$  be a MUPS of the unsatisfiable class  $C_i$ .

1) The MUPS  $K'$  is fully dependent on  $C_i$ , denoted by MUPsf, iff there does not exist any axiom in  $K'$  that contains an unsatisfiable class except  $C_i$ .

2) The MUPS  $K'$  is transitively dependent on  $C_i$ , denoted by MUPSt, iff  $K'$  consists of only one axiom, which contains two unsatisfiable classes  $C_i$  and  $C_j$ , and  $C_j$  is a super class or an equivalent class of  $C_i$ .

3) Otherwise, the MUPS  $K'$  is called uncertainly dependent on  $C_i$ , denoted by MUPSu.

**Definition 3** (directly relevant axiom) Let  $C$  be an atomic OWL class of ontology  $O$ . The set of directly relevant axioms of  $C$ ,  $S(C)$ , is composed of the axioms which are in the form of  $C \subseteq A$  or  $C \equiv A$ .

In the Koala ontology<sup>[7]</sup>, there are three unsatisfiable classes: Quokka, KoalaWithphD and Koala. MUPS(Quokka, Koala) of the unsatisfiable class Quokka is depicted in Fig. 1. Analyzing the directly relevant axioms of the unsatisfiable class Quokka of the MUPS, it can be seen that there is no other unsatisfiable class in it. Hence, the error cannot be fixed by correcting other unsatisfiable classes, in other words, it requires repairing the contradiction of its own definition. It is obvious that an OWL ontology with a significant number of this kind of MUPS is of poor quality.

### The unsatisfiable class:Quokka

Found 1 MUPS

MUPsf 1 MUPSt 0 MUPSu 0

Axioms causing the unsatisfiability: MUPsf MUPS

- 1)  $(\text{Quokka} \subseteq (\exists \text{IsHardWorking} . \{\text{"true"}\langle \text{boolean} \rangle\}))$
- 2)  $\_ (\text{IsHardWorking} \text{ domain } \text{Person})$
- 3)  $\_ (\text{Person} \subseteq \neg \text{Marsupials})$
- 4)  $(\text{Quokka} \subseteq \text{Marsupials})$

Fig. 1 The MUPS of class Quokka obtained by OntoMD

Consider MUPS(KoalaWithphD, koala) of the unsatisfiable class KoalaWithphD depicted in Fig. 2, which contains two labeled regions, each corresponding to a MUPS, denoted by MUPS<sub>1</sub> and MUPS<sub>2</sub>. MUPS<sub>1</sub> is of type MUPSt. Region 1 shows that the unsatisfiable class Koala results in the error of KoalaWithphD. If we fix and repair the error of Koala, MUPS<sub>1</sub> will be NULL. If MUPSt accounts for the vast majority of all MUPS, the ontology is still of relatively high quality. MUPS<sub>2</sub> is a MUPSu on KoalaWithphD. It does not know whether it can give any suggestions on the repair work or not.

### The unsatisfiable class:KoalaWithPhD

Found 2 MUPS

MUPsf 0 MUPSt 1 MUPSu 1

Axioms causing the unsatisfiability: MUPSt MUPS

- 1)  $(\text{KoalaWithPhD} \subseteq \text{Koala})$
- 2)  $\_ (\text{Koala} \subseteq \text{Marsupials})$
- 3)  $(\text{KoalaWithPhD} \subseteq (\exists \text{PhasDegree} . \{\text{PhD}\}))$
- 4)  $\_ (\text{PhasDegree} \text{ domain } \text{Person})$
- 5)  $\_ (\text{Person} \subseteq \neg \text{Marsupials})$

Fig. 2 All MUPS for the unsatisfiable class KoalaWithphD

## 2 All MUPS Identification Based on Discrimination Rules

MIDR aims at identifying all MUPS of the unsatisfiable classes, which is important for explaining the unsatisfiability and the repair work. In section 1.4, MUPS is classified into three types: MUPsf, MUPst and MUPsu. MIDR employs two equivalent transformations and three discrimination rules to obtain them.

### 2.1 Two equivalent transformations

In this subsection, we present two transformations: one is used to split the axioms in a KB into “smaller” axioms to obtain an equivalent KB<sup>[9]</sup>, the other is used to find the cause of a clash between two axioms.

#### 2.1.1 Axiom splitting

The axioms in the following forms:  $C, \neg C, \exists R. C, \forall R. C, \exists R. \{I\}, = n. R, \geq n. R$  and  $\leq n. R$ , are called atomic axioms, otherwise, combination axioms.

For an unsatisfiable class  $C$  w. r. t.  $O$ , there are only two forms of axioms:  $C \sqsubseteq A$  or  $C \equiv A$ , where  $A$  can be an atomic or combination axiom. Obviously, a combination axiom can be split into atomic axioms<sup>[9]</sup>; e. g., in the koala ontology, axiom  $\text{KoalaWithphD} \equiv ((\exists \text{hasDegree. \{phD\}}) \cap \text{Koala})$  can be rewritten into  $\text{KoalaWithphD} \equiv ((\exists \text{hasDegree. \{phD\}}) \sqcap \text{Koala})$ . After the splitting transformation, a split version of ontology  $O$ , denoted by  $O_s$ , is obtained. Instead of including complicated combination axioms,  $O_s$  is composed of atomic axioms, which will help MIDR identify precisely which part of the axioms is responsible for a clash, while ensuring the completeness of the MUPS.

#### 2.1.2 Ontology reduction

Let  $O_s$  be the split version of  $O$ .  $ax_1$  and  $ax_2$  are different axioms in  $O_s$ .  $O_D = O_s - S(C_i)$ .

- The set of unsatisfiable, directly relevant axioms of  $C_i$ , denoted by  $S_{\text{unsatisfy}}(C_i)$ , is composed of the axioms in  $S(C_i)$  that contain an unsatisfiable class except  $C_i$ .

- The set of satisfiable, directly relevant axioms of  $C_i$ ,  $S_{\text{satisfy}}(C_i) = S(C_i) - S_{\text{unsatisfy}}(C_i)$ .

Logical contradiction (semantic defects<sup>[7]</sup>) is fundamentally a clash between two axioms. While Pellet provides support for finding the set of axioms responsible for the unsatisfiability of  $C_i$  in  $O$ , it cannot fix which part of the directly relevant axioms of  $C_i$  cause the clash. Here, we use a reduction transformation to achieve the identification. In the transformation,  $O_s$  is converted equivalently to a set of sub-ontology  $O'_D$ , which is in the form of  $O'_D = O_D \cup \{ax_1\} \cup \{ax_2\}$ , where  $O_D$  represents the ontology which does not contain any axiom in  $S(C_i)$ . If  $C_i$  is still unsatisfiable w. r. t.  $O'_D$ , we can conclude that axioms ( $ax_1, ax_2$ ) cause  $C_i$  unsatisfiable, because there are no other axioms directly relating to  $C_i$  in  $O'_D$ .

Given  $O'_D = O_D \cup \{ax_1\} \cup \{ax_2\}$ , we can use Pellet to obtain the cause (set of axioms) for the clash between  $ax_1$  and  $ax_2$ .

### 2.2 Three discrimination rules

In this subsection, three discrimination rules are presented.

Based on them, MIDR can not only identify all MUPS, but also classify them into three types.

**Rule 1** Let  $ax$  be an axiom in  $O_s$ , if  $ax \in S_{\text{unsatisfy}}(C_i)$  and the right-hand side of  $ax$  is an atomic axiom, then  $ax$  is a MUPst of  $C_i$ .

**Rule 2** If  $C_i$  is still unsatisfiable w. r. t.  $O_D \cup \{ax_1\} \cup \{ax_2\}$ , for  $ax_1, ax_2 \in S_{\text{satisfy}}(C_i)$ , then there is a clash between  $ax_1$  and  $ax_2$ , and the cause for the clash is a MUPsf of  $C_i$ .

A special case:  $S(C_i)$  contains only one axiom  $ax$ , which is in the form  $ax: C_i \sqsubseteq \exists R. C$ , and  $ax \in S_{\text{satisfy}}(C_i)$ . If the filler  $C$  is disjoint from the range of the property  $R$ , then the axiom  $ax$  is a MUPsf of  $C_i$ .

**Rule 3** If there is a clash between  $ax_1$  and  $ax_2$ ,  $ax_1, ax_2 \in S_{\text{unsatisfy}}(C_i)$ , or  $ax_1 \in S_{\text{unsatisfy}}(C_i)$  and  $ax_2 \in S_{\text{satisfy}}(C_i)$ , then the cause for the clash is a MUPsu of  $C_i$  w. r. t. ontology  $O_D \cup \{ax_1\} \cup \{ax_2\}$ .

### 2.3 The procedure of MIDR

In this subsection, we integrate the transformations and discrimination rules mentioned above into MIDR. Given an ontology  $O$ , and the set of the unsatisfiable classes  $\{C_1, C_2, \dots, C_n\}$ , MIDR can obtain all MUPS dependency maps, denoted by  $\text{MAP\_allMUPS}$ . The member in  $\text{MAP\_allMUPS}$  is in the form (key, value), where key is an unsatisfiable class; the corresponding value is the set of all MUPS of the class. The procedure of MIDR is as follows:

- 1) Split ontology  $O$  and obtain the split version of  $O$ :  $O_s$
- 2) For each class  $C_i \in \{C_1, C_2, \dots, C_n\}$  do
- 3) Compute  $S(C_i)$ ,  $S_{\text{satisfy}}(C_i)$  and  $S_{\text{unsatisfy}}(C_i)$
- 4) For each axiom  $ax \in S_{\text{unsatisfy}}(C_i)$ , determine whether  $ax$  is a MUPst of  $C_i$  using discrimination rule 1. Obtain all MUPst of  $C_i$ .
- 5) For each pair of axioms  $ax_1, ax_2 \in S(C_i)$ , if there is a clash between them, determine whether the cause of the clash is a MUPsf (or MUPsu) of  $C_i$  using discrimination rules 2 and 3. Obtain all MUPsf and MUPsu of  $C_i$ .

## 3 Implementation and Evaluation

The MIDR proposed in this paper is implemented in a prototype system OntoMD, which is coded in Java (version 1.5.0.12) and OWL API<sup>[12]</sup>, the latter is a Java interface and is used to represent semantic web ontologies, by which, developers can focus on manipulation of ontologies at a high level of abstraction, and do not need to consider the issues such as parsing and serializing particular syntaxes. Pellet which has been proved to be reliable and stable (version 1.5.0) is employed for unsatisfiability checks. The ontology-centric information stored in SwoopModel<sup>[13]</sup> is used to model ontologies and their associated entities.

The experiment is performed on a PC (Intel Core2 Duo 2.0 GHz), with 1 GB RAM, and 512 MB memory allotted to Java. The test data consist of 10 real-world OWL ontologies, which cover a wide range of unsatisfiable class cases and variation in size, complexity and domain<sup>[9]</sup>. The details of the ontologies are shown in Tab. 1.

To evaluate the performance of our approach, we identify all MUPS of each of the ten ontologies. By comparing them with the output by HST<sup>[10]</sup>, which has been proven to be correct, it can be concluded that MIDR can obtain correct

and complete results (some have different forms). In Tab. 2, we compare the two methods in four aspects.

**Tab. 1** The OWL ontologies used in our experiment

Ontology	Number of classes/ unsatisfiable classes	Description of the ontologies
Koala	20/3	Humans and marsupials
MadCow	54/1	Mad cow disease
Economy	338/51	Economic domain
Tambis	395/144	Biological science
Sweet-jpl	1537/1	Earthscience
University	31/15	Training ontology
Terrorism	100/14	Modeling errors
Transport	445/52	Transportation
BuggyPolicy	15/4	WS-Policies
CHEM-A	48/37	Chemical elements

**Tab. 2** Comparisons of MIDR with HST

Feature	HST	MIDR
Input	An unsatisfiable class; ontology $O$ ; a MUPS	Unsatisfiable class set; ontology $O$
Output	All MUPS of the unsatisfiable class	All MUPS of each unsatisfiable class
Strategy	Enumeration	Three rules
MUPS classification	No	Yes

Tab. 3 gives the number of three types of MUPS (MUPsf, MUPSt, MUPSu) of each ontology respectively. The last column is the total runtime to find all MUPS for all unsatisfiable classes of each ontology. Analyzing the ontologies that have a larger number of unsatisfiable classes: economy, university, transport and CHEM-A, it can be seen that the number of MUPsf is a small fraction of the total number of all MUPS, with the exception of the economy ontology. From a repair point of view, the quality of the economy ontology is poorer than the other three. On the other hand, although CHEM-A contains a large number of MUPS, the number of MUPSt accounts for 90%, so it is still of relatively high quality.

**Tab. 3** The results of MIDR on the test data

Ontology	Number of MUPS			Total time/s
	MUPsf	MUPSt	MUPSu	
Koala	2	1	1	0.327
MadCow	1	0	0	0.204
Economy	35	17	11	8.932
Tambis	121	223	55	765.129
Sweet-jpl	1	0	0	0.719
University	7	18	3	1.316
Terrorism	5	9	0	1.189
Transport	24	47	8	140.55
BuggyPolicy	3	2	1	1.28
CHEM-A	4	53	2	1.578

The process of debugging an ontology with numerous MUPS can be conducted from a MUPS-driven view. Modelers can fix and repair MUPsf first, which will automatically reduce the number of MUPSt, expose MUPSu as MUPsf or NULL, and make some of the unsatisfiable classes satisfiable. The process works iteratively, only considering MUPsf

in each iteration.

## 4 Conclusion and Future Work

In this paper, we present a MUPS identification approach named MIDR based on discrimination rules, which aims at obtaining all MUPS of an ontology and classifying them into three types. The experiments on ten ontologies show that MIDR can achieve correct and complete results. The classification results obtained by MIDR can be used to evaluate the quality of an ontology from a repair point of view, and help modelers design a much more efficient proposal for the repair work.

In future work, we will improve MIDR in two ways: 1) Decrease the computational complexity by using more analytical techniques for finding the causes of clashes between two axioms; 2) Generate output reports in the form of natural language in order to help users understand the meaning of MUPS easily.

## References

- [1] Schlobach S, Cornet R. Non-standard reasoning services for the debugging of description logic terminologies [C]//*Proc of the 8th International Joint Conference on Artificial Intelligence*. Acapulco, Mexico, 2003: 335 – 362.
- [2] Haase P, van Harmelen F, Huang Zhisheng, et al. A framework for handling inconsistency in changing ontologies [C]//*Proc of the 4th International Semantic Web Conference*. Berlin: Springer-Verlag, 2005: 353 – 367.
- [3] Schlobach S, Huang Zhisheng. Inconsistent ontology diagnosis: framework and prototype [R]. Karlsruhe, Germany: SEKT Project, 2003.
- [4] Wang H, Horridge M, Rector A, et al. Debugging OWL DL ontologies: a heuristic approach [C]//*Proc of the 4th International Semantic Web Conference*. Berlin: Springer-Verlag, 2005: 745 – 757.
- [5] Schlobach S, Huang Zhisheng, Ronald C, et al. Debugging incoherent terminologies [J]. *Journal of Automated Reasoning*, 2007, **39**(3): 317 – 349.
- [6] Parsia B, Sirin E, Kalyanpur A. Debugging owl ontologies [C]//*Proc of the 14th International World Wide Web Conference*. New York: ACM Press, 2005: 633 – 640.
- [7] Kalyanpur A, Parsia B, Sirin E, et al. Debugging unsatisfiable classes in OWL ontologies [J]. *Journal of Web Semantics*, 2005, **3**(4): 268 – 293.
- [8] Meyer T, Lee K, Booth R, et al. Finding maximally satisfiable terminologies for the description logic ALC [C]//*Proc of the 21st National Conference on Artificial Intelligence*. Portland: AAAI Press, 2006: 269 – 274.
- [9] Kalyanpur A. Debugging and repair of OWL ontologies [D]. College Park: University of Maryland, College Park, 2006.
- [10] Kalyanpur A, Parsia B, Horridge M, et al. Finding all justifications of OWL DL entailments [C]//*Proc of the 6th International Semantic Web Conference*. Berlin: Springer-Verlag, 2007: 267 – 280.
- [11] Horrocks I, Patel-Schneider P F. Reducing OWL entailment to description logic satisfiability [J]. *Journal of Web Semantics*, 2004, **1**(4): 345 – 357.
- [12] Bechhofer S, Lord P, Volz R. Cooking the semantic web with the OWL API [C]//*Proc of the Second International Semantic Web Conference*. Berlin: Springer-Verlag, 2003: 659 – 675.
- [13] Kalyanpur A, Parsia B, Sirin E, et al. Swoop: a web ontology editing browser [J]. *Journal of Web Semantics*, 2006, **4**(2): 144 – 153.

# 基于判别规则的最小不一致知识子集识别

王文斌 杨帆 饶一梅 陈秋双

(南开大学信息技术科学学院, 天津 300071)

**摘要:**为解释本体中概念不满足的原因,利用 2 个对等转换(即公理细化和本体约减)与 3 个判别规则识别不满足概念  $C$  的最小不一致知识子集(MUPS). 其中,判别规则基于不满足概念的传递性,将 MUPS 分为 3 类:完全依赖于  $C$ (MUPS<sub>f</sub>)、传递依赖于  $C$ (MUPS<sub>t</sub>) 和不确定依赖于  $C$ (MUPS<sub>u</sub>). 实验结果表明:在本体不满足概念的 MUPS 中,MUPS<sub>t</sub> 往往占大多数,但只有 MUPS<sub>f</sub> 可以明确指出概念不满足的根本原因. 本体建模人员和领域专家可以采用迭代修复方式,每一次修复只考虑 MUPS<sub>f</sub>,以提高修复效率. 所得分类结果对于从修复角度评价本体质量以及指导修复工作都具有重要意义.

**关键词:**本体调试;最小不一致知识子集;判别规则

**中图分类号:**TP391