本体推理机求解 Mups 的性能评测研究

欧阳丹彤 张 瑜 叶育鑫
(吉林大学计算机科学与技术学院 长春 130012)
(吉林大学符号计算与知识工程教育部重点实验室 长春 130012)

摘 要 求解极小不可满足保持子术语集(Mups)是不一致术语集调试的核心工作.在构建术语集依赖关系图模型基础上,从概念之间的依赖关系角度出发,定义语义依赖度、语义簇、依赖度分布 3 个指标反映本体术语集的复杂程度;通过讨论不可满足概念数目、冲突公理集基数和冲突公理基数对 Mups 问题求解难易的影响,定义冲突公理集最大基数和冲突公理最大基数两个指标反映不一致本体术语集的数据复杂程度;基于这些复杂性指标,设计针对 Mups 问题的不一致本体数据标准测试集(Mups Benchmark,MupsBen)来评测 Pellet、Hermit、FaCT++、JFact 和 TrOWL 这 5 种推理机在黑盒算法下求解 Mups 的性能.评测实验显示,所定义的复杂度指标能够有效反映 Mups 求解问题的数据复杂程度.对于特定推理机,其性能随测试数据的结构复杂程度的增大而降低;对于不同推理机,由于其内在推理机制与优化策略的差别,在不同复杂度指标下表现出不同的性能差异.

关键词 不一致术语集;极小不可满足保持子术语集;标准检查程序;MupsBen;人工智能
中图法分类号 TP18 **DOI 号** 10.11897/SP.J.1016.2017.01422

Research on Evaluating Ontology Reasoners for Calculating Mups

OUYANG Dan-Tong ZHANG Yu YE Yu-Xin
(*School of Computer Science and Technology, Jilin University, Changchun* 130012)
(*Key Laboratory of Symbolic Computation and Knowledge Engineering of Ministry of Education, Changchun* 130012)

Abstract Calculating minimal unsatisfiability preserving sub-terminologies (Mups) plays a crucial role in debugging incoherent terminology. On the basis of terminology dependency graph model and from the perspective of dependences between concepts, we define three metrics of semantic dependence, semantic cluster and dependence distribution to reflect the complexity of terminology. By studying unsatisfiable concept number, conflict axiom set cardinal and conflict axiom cardinal how to influence the difficulty of calculating Mups, we define two metrics of max cardinal of conflict axiom set and max cardinal of conflict axiom to reflect the complexity of incoherent terminology. Then, we devise a Mups Benchmark (MupsBen) to evaluate the performances of reasoners for calculating Mups based on those metrics. MupsBen is able to generate incoherent terminologies to evaluate the performances of Pellet, Hermit, FaCT++, JFact and TrOWL by black-box method. Evaluation experiments show that the proposed metrics can effectively reflect the complexity of Benchmark data for calculating Mups. The performance of specific reasoner decreases with the increase of complexity of test data. Because of the differences of inference mechanisms and optimization strategies, the performances of various reasoners have quite differences for diverse complexity metrics.

Keywords incoherent terminology; minimal unsatisfiability preserving sub-terminologies; benchmark; MupsBen; artificiall intelligence

收稿日期:2016-03-31;在线出版日期:2016-08-30.本课题得到国家自然科学基金(61133011,41172294,61170092,61272208)、吉林省科技发展计划(201201011)资助.**欧阳丹彤**,女,1968 年生,博士,教授,博士生导师,中国计算机学会(CCF)高级会员,主要研究领域为基于模型诊断、自动推理.E-mail:ouyd@jlu.edu.cn.**张 瑜**,男,1982 年生,博士研究生,主要研究方向为描述逻辑、本体调试.**叶育鑫**(通信作者),男,1981 年生,博士,副教授,主要研究方向为语义 Web、本体工程.E-mail:yeyx@jlu.edu.cn.

1 引　言

本体可以实现信息共享和重用领域知识，因此被广泛地应用于语义 Web 技术[1-2]. 围绕知识库与本体的查询、推理、融合、匹配与构建等一系列研究工作也相继展开. 其中，研究上述问题的有效手段之一是：通过分析本体数据的结构特征和复杂程度来评测解决方案的可用性和适用性. 在知识库系统的评测方面，Lehigh 大学的 SWAT 研究小组开发了用于评测知识库推理能力的 LUBM(Lehigh University Benchmark)本体数据标准测试集[3]，Ma 等人[4]指出 LUBM 生成的本体数据各个类之间的关联度不足，于是在类与类之间添加了更多的属性关联，将 LUBM 扩展成 UOBM(University Ontology Benchmark). 由于 LUBM 和 UOBM 生成的都是单个本体数据，Benchmark 测试数据的复杂性的提高受到较大的限制，于是 Li 等人[5]开发了多本体合成的 MOSB (Multi-ontology Synthetic Benchmark)，从而大大提高了 Benchmark 测试数据在评测知识库系统方面的适用性. Imprialou 等人[6]则着眼于本体查询问题，设计一个测试查询生成器，该生成器能够生成与输入本体相关的测试查询，用来验证待测系统能否正确地完成对应的推理任务. 在本体融合任务中，Raunich 等人[7]的工作具有代表性，该工作设计了输入本体的覆盖度、融合结果的紧致性以及合成本体的冗余度 3 个度量指标，在不同规模和复杂程度的 5 种测试场景中，评测本体融合方法的性能质量. 在本体匹配领域中，最新的代表性成果有 Ferrara 等人[8]开发的语义网实例生成器(SWING)，SWING 能够半自动地生成一系列测试案例，每个测试案例都对应一组实例断言. SWING 所评测的是单语种本体匹配器，这与 Meilicke[9]的工作不同，后者所针对的是多语种本体匹配器，它使用了来自本体对齐评测项目(OAEI)的 OntoFarm 数据集并将其翻译成 8 种不同语种的本体并设定这些本体之间的对齐关系. Rosoiu[10]的工作也与 OAEI 相关，不同之处在于 Rosoiu 首先指出了 OAEI 存在的缺陷，为了克服这些缺陷，开发了不依赖种子本体，并且允许调整输入参数来覆盖问题空间的模块化测试生成器.

在本体构建过程中，由于建模者领域知识的缺乏和建模经验的不足，很容易出现本体不一致的情况，本体不一致会导致从本体中推出相互矛盾的知识，体现在术语集中则是出现了不可满足概念，它表明本体中概念的形式化定义出现了逻辑错误，为了确保本体质量和推理效率，有必要对这类错误进行诊断并处理. 因此如何通过有效方法获取本体中不可满足概念的冲突术语集，是达到本体调试目的的最主要手段. 研究者们已经展开各种方法的研究，其中包括 Meyer 等人[11]的极大可满足概念调试方法、Schlobach 等人[12]的模型诊断方法和 Kalyanpur 等人[13-15]的辩解(Justification)碰集计算方法等等. 在本体调试过程中，辩解是本体中保持某一蕴含的最小子本体，Bail 等人[16]发现，在很多本体中，形式上不同的辩解却存在着同构现象，依此定义了子表达式同构和引理同构这两种类型，用来减化本体中的辩解. 为了提高大规模本体中概念的分类效率，Aslani 等人[17]设计了并行 TBox 概念分类器的算法结构，使用并行线程共享统一内存，并证明了提出的并行分类算法的健全性和完备性. Bail 等人和 Aslani 等人的工作都是建立在完备推理机基础之上的，Stoilos[18]则着眼于不完备推理机，采取的方法是基于给定的查询 q，TBox $\mathcal{T}$ 和一个不完备的推理机计算出来一个 TBox 修正，该修正能够帮助推理机获得一个完备的查询回答，因而能够改善这类推理机的完备性又不以牺牲它们的性能为代价.

所谓模型诊断的方法，是将术语集看作“系统”，将术语集中的公理看作“部件”，“系统描述”指该术语集是一致的，但是在“观察”中发现术语集中存在不可满足概念，于是 Schlobach 等人[12]基于模型诊断的极小冲突集提出了极小不可满足保持子术语集(Minimal Unsatisfiability Preserving Sub-TBoxes，Mups)，并提供一个调用外部推理机支持的黑盒算法和基于 tableau 演算的白盒算法来求解 Mups. 相比较而言，白盒算法在针对特定描述逻辑语言进行优化时效率较高，但需要根据特定的描述逻辑语言构件来选择特定推理机进行修改，可移植性较差，适用范围较窄. 黑盒算法由于直接调用外部推理机，不需要关注推理机内部实现细节，因而可移植性强，适用范围广，并且无须限定推理机的类型，因而可以充分利用推理机的能力. 上述方法从不同方面和程度上解决了不同情形下的本体推理问题，但对本体调试问题，尤其是 Mups 问题的方法评测工作尚未展开. 因此，在本体调试领域，国内外所开展的研究中，尚未涉及到针对 Mups 求解问题设计一个评测推理机性能的 Benchmark 这一方面的研究. 本文通过分析影响 Mups 问题求解效率的本体数据的因素，给出不同复杂程度的本体测试数据生成方法，并利用

生成的测试数据评测基于黑盒的求解算法.基于黑盒求解算法的实质是通过计算本体术语集子集的一致性来发现不可满足概念的 Mups.其中,一致性的验证工作是通过调用本体推理机中的标准推理任务实现的.目前主流的本体推理机,如 Pellet[19]、Hermit[20]、FaCT++[21]和 TrOWL[22]以及在 FaCT++基础之上开发出 JFact,均支持该任务.本文考察不同推理机在求解 Mups 问题上的适用性.通过生成复杂程度不同的本体测试数据(即 Benchmark),测试不同推理机的 Mups 问题的求解效率.

2 不一致术语集复杂性分析

本体是某个领域内的概念以及概念之间关系的规范化说明.描述逻辑是表示本体的形式化语言,用描述逻辑表示的本体知识库由 TBox 和 ABox 两部分构成.TBox 也叫做术语集(terminology),用$\mathcal{T}$表示,它包括概念和角色,体现为 $C\sqsubseteq D$ 或 $C\equiv D$ 两种形式的公理.ABox 是关于领域内个体的断言集(assertion),包括概念断言 $C(a)$和角色断言 $R(a,b)$.

例 1. 若有术语集$\mathcal{T}$如下:

$\alpha_1: father_{(S_1)}\equiv \forall haschild.human_{(B_1)}$;

$\alpha_2: parent_{(S_2)}\equiv father_{(S_1)}\sqcup mother_{(B_2)}$;

$\alpha_3: grandmother_{(S_3)}\sqsubseteq parent_{(S_2)}\sqcap woman_{(B_3)}$;

$\alpha_4: grandfather_{(S_4)}\sqsubseteq \exists haschild.father_{(S_1)}\sqcap parent_{(S_2)}$;

$\alpha_5: grandparent_{(S_5)}\equiv grandmother_{(S_3)}\sqcup grandfather_{(S_4)}$;

$\alpha_6: wife_{(S_6)}\sqsubseteq \exists hasdaughter.girl_{(B_4)}\sqcap hashusband.man_{(B_5)}$;

$\alpha_7: police_{(S_7)}\sqsubseteq man_{(B_5)}$;

$\alpha_8: policewoman_{(S_8)}\sqsubseteq wife_{(S_6)}\sqcap police_{(S_7)}$;

$\alpha_9: student_{(C_1)}\equiv boy_{(A_1)}\sqcap \neg boy_{(\neg A_1)}$;

$\alpha_{10}: undergraduate_{(C_2)}\equiv \exists hasCourse.Computer_{(A_2)}\sqcap \forall hasCourse.\neg Computer_{(\neg A_2)}$;

$\alpha_{11}: graduate_{(C_3)}\sqsubseteq student_{(C_1)}\sqcap undergraduate_{(C_2)}$;

$\alpha_{12}: teacher_{(C_4)}\sqsubseteq professor_{(A_3)}\sqcap \neg professor_{(\neg A_3)}$;

$\alpha_{13}: faculty_{(C_5)}\sqsubseteq graduate_{(C_3)}\sqcap teacher_{(C_4)}$.

该术语集由 13 个公理组成,包括 24 个概念和 3 个角色,右下角的标号是概念名的替代符,其中 α_1、α_2、α_5、α_9和 α_{10}是形如 $C\equiv D$ 的等价公理,其他是形如 $C\sqsubseteq D$ 的包含公理.

2.1 求解 Mups 问题的黑盒算法

如果术语集$\mathcal{T}$中的某个概念 C,存在一个解释$\mathcal{I}$使得 $C^{\mathcal{I}}$非空,则 C 相对于$\mathcal{T}$是可满足的,否则是不可满足的,导致概念不可满足的原因是同时存在着两个互补或不相交的父类,这样的概念是没有意义的,这成为本体调试工作需要解决的根本性逻辑错误.因而文献[12]将本体的不一致(incoherence)解释为术语集中存在不可满足概念,定义如下.

定义 1. 不一致(incoherent)术语集[12].术语集$\mathcal{T}$是不一致的当且仅当$\mathcal{T}$中存在一个不可满足概念.

对于例 1 的术语集$\mathcal{T}$,可以发现公理 α_9所定义的概念 $student_{(C_1)}$是不可满足的,原因是 $student_{(C_1)}$被定义为概念 $boy_{(A_1)}$与其补集的交,同理可知 α_{10}所定义的 $undergraduate_{(C_2)}$也是不可满足的.而 α_{11}中 $graduate_{(C_3)}$不可满足是因为将其定义为两个不可满足概念 $student_{(C_1)}$和 $undergraduate_{(C_2)}$的子集.同样可知 $teacher_{(C_4)}$和 $faculty_{(C_5)}$也是不可满足的.

基于描述逻辑系统的调试目标是找出那些存在逻辑冲突的公理集,它类似于命题逻辑中的极小冲突集,修改或删除该公理集就可以确保原本不可满足的概念变得可满足.基于这一目标,Schlobach 等人提出了极小不可满足保持子术语集(Mups).

定义 2. Mups[12,23].设 C 是术语集$\mathcal{T}$的一个不可满足概念,对于$\mathcal{T}$的某个子集$\mathcal{T}'$,如果 C 在$\mathcal{T}'$中不可满足,而在$\mathcal{T}'$的任意真子集$\mathcal{T}''$中都可满足,则$\mathcal{T}'$是 C 的一个 Mups.

可以求出例 1 的不一致术语集$\mathcal{T}$中不可满足概念 $student_{(C_1)}$, $undergraduate_{(C_2)}$, $graduate_{(C_3)}$, $teacher_{(C_4)}$, $faculty_{(C_5)}$的 Mups 如下:

$\mathrm{Mups}(C_1)=\{\{\alpha_9\}\}$; $\mathrm{Mups}(C_2)=\{\{\alpha_{10}\}\}$;

$\mathrm{Mups}(C_3)=\{\{\alpha_9,\alpha_{11}\},\{\alpha_{10},\alpha_{11}\}\}$;

$\mathrm{Mups}(C_4)=\{\{\alpha_{12}\}\}$;

$\mathrm{Mups}(C_5)=\{\{\alpha_9,\alpha_{11},\alpha_{13}\},\{\alpha_{10},\alpha_{11},\alpha_{13}\},\{\alpha_{12},\alpha_{13}\}\}$.

比较以上 5 个概念的 Mups,对于 C_1, C_2, C_4来说,它的不可满足性是由于自身定义出现了逻辑冲突,因而它的 Mups 就是自身所属的公理.然而,对于 C_3来说,它的不可满足性是由 C_1和 C_2两个不可满足概念引起的,所以,为了找出 C_3的极小冲突公理集,必须首先找到 C_1和 C_2的极小冲突公理集,因此,求解 C_3的 Mups 的复杂程度要大于 C_1和 C_2.而 C_5的不可满足性与 C_3、C_4都有关系,因此求解 C_5的 Mups 自然要复杂于 C_3和 C_4.基于这一认识,提出了冲突公理集基数和冲突公理基数两个表征 Mups 复杂程度的指标.

定义 3. 冲突公理集基数.对于术语集中的不可满足概念,造成其不可满足的冲突公理集的个数,称为冲突公理集基数.

设不可满足概念 C 的冲突公理集基数为$\mathcal{N}(C)$，对于例 1 中的不可满足概念，则有

$\mathcal{N}(C_1)=\mathcal{N}(C_2)=\mathcal{N}(C_4)=1, \mathcal{N}(C_3)=2, \mathcal{N}(C_5)=3.$

定义 4.　冲突公理基数. 对于术语集中的不可满足概念，造成其不可满足的每一组冲突公理集内公理的数目，称为冲突公理基数.

设不可满足概念 C 的冲突公理基数为$\mathcal{L}(C)$，对于例 1 中的不可满足概念，则有

$\mathcal{L}(C_1)=\mathcal{L}(C_2)=\mathcal{L}(C_4)=\{1\}$, $\mathcal{L}(C_3)=\{2,2\}$,
$\mathcal{L}(C_5)=\{3,3,2\}$.

进一步地，定义 C 的冲突公理最大基数为冲突公理集合里的元素的最大值，即$\mathcal{L}_{\max}(C)=\max(\mathcal{L}(C))$. 则有

$\mathcal{L}_{\max}(C_1)=\mathcal{L}_{\max}(C_2)=\mathcal{L}_{\max}(C_4)=\max(\{1\})=1$,
$\mathcal{L}_{\max}(C_3)=\max(\{2,2\})=2$,
$\mathcal{L}_{\max}(C_5)=\max(\{3,3,2\})=3$.

黑盒算法在调用推理机求解问题的过程中，每一次寻找不可满足概念 C 的极小冲突公理集时，都需要对整个术语集执行扩张和收缩两个操作. 在扩张阶段，算法首先生成一个空集 K，然后从术语集往 K 中逐个添加公理，直到待求解的概念 C 变得不可满足. 在收缩阶段中，算法逐项地从 K 中删除公理来压缩求解空间，每次删除一个公理都要对 C 进行一次可满足性测试. 若删除某个公理后，C 变得可满足，证明该公理是造成 C 不可满足的原因，它属于 C 的极小冲突公理集；若删除某个公理后，C 仍然不可满足，说明该公理对 C 的不可满足性不产生影响，删除即可. 反复执行上述操作，直到遍历整个集合 K，就能得出极小冲突公理集.

黑盒法的局限性在于它不一定能够求出全部的 Mups. 于是 Kalyanpur[24] 对黑盒法进行了改进，通过扩展和压缩过程对依赖集进行过滤，并将基于模型诊断的碰集方法引入其中，提出了 MUPSHST 算法，能够求解出全部的 Mups.

2.2　Mups 问题的术语集结构复杂性分析

由于本体构建专家是基于某一特定的领域知识进行本体建模的，因此，领域不同导致构建出的本体差异很大. 然而，在本体构建过程中，无论针对哪一领域，都需要首先为该领域知识建立本体模型. 在建立的本体模型中，概念与概念之间存在着复杂的依赖关系.

如果将本体的概念看做图的节点，概念与概念之间的依赖关系看做图的边，就可以将本体模型抽象为一个图模型.

针对本体中的术语集，定义术语集的依赖关系图模型如下.

定义 5.　依赖关系图. 一个术语集$\mathcal{T}$的依赖关系图是一个有向图 $G=(V,E)$，其中 V 是节点集，表示$\mathcal{T}$中概念的集合，E 是边集，表示概念之间关系的集合.

由于将术语集中概念之间的依赖关系表示为依赖关系图模型，因此，通过对依赖关系图的复杂性分析可以发现影响术语集复杂程度的因素.

将例 1 的术语集$\mathcal{T}$表示为依赖关系图模型（如图 1 所示），图中的节点表示概念，有向弧表示概念之间存在的依赖关系.

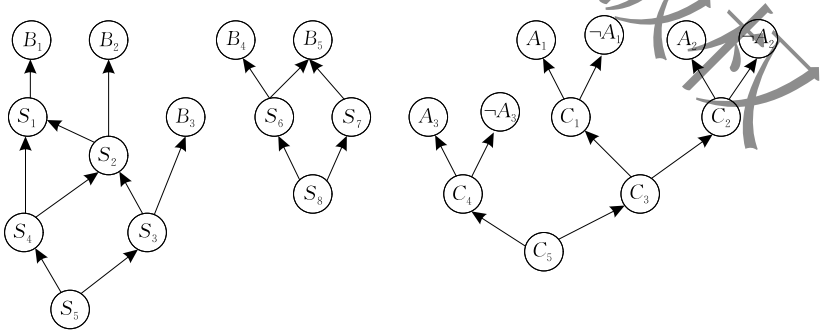


图 1　术语集$\mathcal{T}$的依赖关系图模型

概念和关系是依赖关系图模型的基本元素，在依赖关系图中，每个关系都会将一对概念联系起来. 依赖关系图中的原子概念是指没有父概念的那些概念，也可以叫做根概念，因为其他概念都是从它们继承下来的. 概念的依赖路径是指从该概念出发抵达原子概念所经过的依赖关系，它反映了该概念与根概念间的语义距离. 图 1 的依赖图有 24 个概念，10 个原子概念，S_5-S_3-S_2-S_1-B_1 是从 S_5 到根概念 B_1 的一条路径，路径长度为 4.

当黑盒算法调用概念的路径越长外部推理机进行可满足性检测时，它在依赖关系图层次结构中就越处于下层，它所继承的祖先概念就越多. 当推理机

检测这类概念时,就会沿着概念的路径一直上溯到根概念,在上溯过程中途经的所有概念,都会对其进行可满足性检测,只有在所有的祖先节点的可满足性检测完成后,才能最后确定该概念的可满足性,因而,会耗费推理机大量的推理时间,这反映出了概念的复杂程度对推理机性能具有重要的影响.

从推理机角度出发,当黑盒算法调用推理机求解每个不可满足概念的 Mups 时,实际上就是去寻找造成该概念不可满足性的所有极小冲突公理集.因此,Mups 的复杂性决定于不可满足概念所继承的父概念的 Mups 的复杂性,在依赖图层次结构中越处于下层的不可满足概念,它的 Mups 的复杂度就越高.

3 不一致术语集复杂度指标

基于术语集依赖关系图模型和 Mups 的复杂性分析,给出了表征术语集复杂度的指标.

3.1 语义依赖度

将术语集表示为依赖关系图模型之后,对依赖关系图的复杂性分析就能够发现影响术语集复杂程度的因素.为了量化依赖关系图模型中概念节点之间的依赖关系,定义术语集中概念的语义依赖度如下.

定义 6. 概念的语义依赖度.在术语集$\mathcal{T}$的依赖关系图中,从某一概念出发,经过一系列依赖关系到达原子概念所经过的最多概念个数,称为该概念的语义依赖度.其中原子概念特指没有父概念的那些概念.概念 C 的语义依赖度 $sd(C)$可以递归定义如下

若C是原子概念,则$sd(C)=0$;

若$C\doteq C_1$,则$sd(C)=sd(C_1)$;

若$C\doteq\neg C_1$,则$sd(C)=sd(C_1)$;

若$C\doteq\exists r.C_1$,则$sd(C)=sd(C_1)+1$;

若$C\doteq\forall r.C_1$,则$sd(C)=sd(C_1)+1$;

若$C\doteq C_1\sqcap C_2$,则$sd(C)=\max(sd(C_1),sd(C_2))+1$;

若$C\doteq C_1\sqcup C_2$,则$sd(C)=\max(sd(C_1),sd(C_2))+1$;

其中$\doteq$表示"$=$"或"$\sqsubseteq$".

称语义依赖度为 1 的概念为简单概念,否则为复杂概念.

若术语集$\mathcal{T}$包括 n 个概念,则术语集的平均语义依赖度为

$$\bar{\lambda}=\frac{\sum_{i=1}^{n}sd(C_i)}{n} \tag{1}$$

术语集的最大语义依赖度可定义为

$$\lambda=\max(sd(C_i)),\ 1\leq i\leq n \tag{2}$$

3.2 语义簇与聚合度

在图 1 所描绘的依赖关系图模型中,某一部分概念之间发生了聚类现象,出现了 3 个概念簇,概念簇内部的概念之间存在着紧密的关联,而概念簇与概念簇之间则相互独立.针对这一现象,提出了语义簇这一表征术语集内部结构特征的指标.

定义 7. 语义簇.术语集$\mathcal{T}$中,从简单概念出发,经由依赖关系所连接的概念构成的子术语集$\mathcal{T}'\subseteq\mathcal{T}$称为该术语集的一个语义簇.

语义簇中包括的概念数目称为该语义簇的直径.

设术语集的概念总数为 n,语义簇数目为 k,第 j 个语义簇的直径为 d_j,不属于任意一个语义簇的简单概念数目为 m,则术语集的语义簇限定条件为

$$m+\sum_{j=1}^{k}d_j=n \tag{3}$$

进一步地,假设术语集中某个概念 C_i 的语义依赖度为 $sd(C_i)$,最大语义依赖度为 λ,则术语集的依赖聚合度定义为

$$\eta=\frac{\sum_{j=1}^{k}\sum_{i=1}^{d_j}sd(C_i)}{\lambda n} \tag{4}$$

当 $m=n$ 时,根据式(3),$k=0$,表明术语集中不存在语义簇,此时依赖聚集度取得最小值 $\eta_{\min}=0$;当 $m=0$ 时,根据式(3),有$\sum_{j=1}^{k}d_j=n$,表明术语集中不存在任何非语义簇的简单概念,此时依赖聚合度取得最大值

$$\eta_{\max}=\frac{\sum_{j=1}^{k}\sum_{i=1}^{d_j}sd(C_i)}{\lambda\sum_{j=1}^{k}d_j} \tag{5}$$

3.3 依赖度分布

由语义依赖关系生成的术语集可知,在同一个语义簇内部存在着概念之间的依赖,假设第 j 个语义簇的最大语义依赖度为λ_j,则依赖度分布定义如下

$$p_j(C_i)=i,\quad i\in[1,\lambda_j] \tag{6}$$

这表明,语义簇内部概念的语义依赖度呈斜率为 1 的线性分布,此时,

$$\sum_{i=1}^{d_j}sd(C_i)=\frac{d_j(1+d_j)}{2} \tag{7}$$

并且语义簇的直径与该语义簇的最大语义依赖度满足 $d_j=\lambda_j$.

将式(7)代入式(5)可得

$$\eta_{\max}=\frac{\sum_{j=1}^{k}d_j(1+d_j)}{2\lambda\sum_{j=1}^{k}d_j} \tag{8}$$

考虑语义簇之间的依赖度满足均匀分布，则 k 个语义簇的直径均相同，设直径为 d，又由于 $d_j=\lambda_j$，则有 $d=\lambda$，这时式(8)可转化为

$$\eta_{\max}=\frac{kd(1+d)}{2\lambda kd}=\frac{1+d}{2\lambda}=\frac{1+\lambda}{2\lambda} \tag{9}$$

又因为依赖聚合度取最大值时，$m=0$，由式(3)可得

$$\sum_{j=1}^{k}d_j=n\Leftrightarrow kd=n\Leftrightarrow k\lambda=n \tag{10}$$

所以最大依赖聚合度与概念总数，语义簇个数和最大语义依赖度有关，因此可得

$$\eta_{\max}=\frac{1+\lambda}{2\lambda} \tag{11}$$

或由式(10)得

$$\eta_{\max}=\frac{k+n}{2n} \tag{12}$$

3.4　冲突公理集最大基数

根据定义 3，不一致术语集中不可满足概念的冲突公理集基数越大，造成该概念不可满足性的极小冲突公理集就越多，而黑盒算法需要调用推理机找出所有的极小冲突公理集，推理机需要耗费的时间自然越多.

定义 8.　冲突公理集最大基数. 设不一致术语集 $\mathcal{T}$ 中，存在 u 个不可满足概念，若第 i 个不可满足概念的冲突公理集基数为 $\mathcal{N}(C_i)$，则 $\mathcal{T}$ 的冲突公理集最大基数为

$$\mathcal{N}_{\max}(\mathcal{T})=\max(\mathcal{N}(C_i)),\ 1\leq i\leq u \tag{13}$$

对于例 1 中的不可满足概念，由定义 3 可知，

$\mathcal{N}(C_1)=\mathcal{N}(C_2)=\mathcal{N}(C_4)=1,\mathcal{N}(C_3)=2,\mathcal{N}(C_5)=3.$

则 $\mathcal{N}_{\max}(\mathcal{T})=3$.

3.5　冲突公理最大基数

根据定义 4，不可满足概念的冲突公理最大基数越大，表明从根不可满足概念到该不可满足概念所经历的依赖关系和派生概念就越多，在推理机求解 Mups 的时候，涉及到的不可满足概念就越多，这些都影响着推理机求解 Mups 的效率.

定义 9.　冲突公理最大基数. 设不一致术语集 $\mathcal{T}$ 中，存在 u 个不可满足概念，若第 i 个不可满足概念的冲突公理最大基数为 $\mathcal{L}_{\max}(C_i)$，则 $\mathcal{T}$ 的冲突公理集最大基数为

$$\mathcal{L}_{\max}(\mathcal{T})=\max(\mathcal{L}_{\max}(C_i)),\ 1\leq i\leq u \tag{14}$$

对于例 1 中的不可满足概念，由定义 4 可知，

$\mathcal{L}_{\max}(C_1)=\mathcal{L}_{\max}(C_2)=\mathcal{L}_{\max}(C_3)=\max(\{1\})=1,$

$\mathcal{L}_{\max}(C_3)=\max(\{2,2\})=2,$

$\mathcal{L}_{\max}(C_5)=\max(\{3,3,2\})=3.$

则 $\mathcal{L}_{\max}(\mathcal{T})=3$.

进一步地分析发现，术语集的冲突公理最大基数与它的最大语义依赖度可相互转换. 这表明，冲突公理基数越大，在依赖关系图的依赖路径上与它具有依赖关系的其他不可满足概念也就越多，表现出来就是语义依赖度越大.

4　MupsBen 设计方案

MupsBen 的设计是建立在对术语集复杂性分析的基础之上的，它在术语集复杂度指标的指导下，生成不同复杂度的术语集.

由于术语集中的概念一部分是可满足的，另一部分是不可满足的，因此可满足概念生成器和不可满足概念生成器需要分开设计. 设计可满足概念生成器时，将可满足概念数目、最大语义依赖度，语义簇数目这 3 个指标作为构造参数，生成可满足概念子术语集. 设计不可满足概念生成器时，由于术语集的冲突公理集最大基数与它的最大语义依赖度可相互转换，所以只需两者取一即可，因此将不可满足概念数目、冲突公理集最大基数和冲突公理最大基数这 3 个指标作为构造参数，生成不可满足概念子术语集. 将两个子术语集合并就能得到所要生成的不一致术语集. 生成器所生成的不一致术语集限定在非循环 $\mathcal{ALC}$ 语言范畴内，该语言遵循如下语法规则：

$$C,D\rightarrow\top\mid\bot\mid A\mid\neg C\mid C\sqcap D\mid C\sqcup D\mid\exists r.C\mid\forall r.C,$$

其中，A 是原子概念，r 是原子角色，C、D 是概念描述.

设计 MupsBen 的可满足概念部分时，原子概念使用符号 $B_1,B_2,\cdots,B_m$ 表示，原子角色使用符号 $r_1,r_1,\cdots,r_n$ 表示，请参考表 1(可满足概念生成器构造部件). 设计它的不可满足概念部分时，原子概念使用符号 $A_1,A_2,\cdots,A_p$，原子角色使用符号 $t_1,t_1,\cdots,t_q$ 表示，请参考表 2(不可满足概念生成器构造部件). 由于原子概念和原子角色表现的是某一领域内最一般最基本的知识元，所以，在原子概念和原子角色之间引入依赖关系，就可以在它们基础之上构建

出复杂的、表达意义更丰富的概念.因此,MupsBen所构建出的本体内的概念和角色都是变量符号的表示形式.

4.1 可满足概念生成器

可满足概念的构造部件包括构造算子和操作数两部分(如表1所列).在可满足概念生成器中,构造算子特指⊓、⊔、∃、∀、⊑和≡这6种,操作数则指原子概念和原子角色两种.

表1 可满足概念生成器构造部件

Constructor table		Syntax
seTab	*subClassOf*	$S_1 \sqsubseteq S_2$
	equivalentClass	$S_1 \equiv S_2$
iuTab	*intersectionOf*	$S_1 \sqcap S_2$
	unionOf	$S_1 \sqcup S_2$
asTab	*allValuesFrom*	$\forall r_1.B_1$
	someValuesFrom	$\exists r_1.B_1$
Operand table		
atomSet	$B_1, B_2, \cdots, B_m$	
roleSet	$r_1, r_1, \cdots, r_n$	

根据构造算子的种类不同,构造算子表Constructor table可划分为3个子表,*seTab*子表存放包含关系(⊑)和等价关系(≡)两种算子,*iuTab*子表存放合取(⊓)和析取(⊔)两种算子,*asTab*子表存放全称量词(∀)和存在量词(∃)两种算子.

根据操作数的种类不同,操作数表Operand table可划分为两类操作数集合,*atomSet*集合中存放的是用作种子的原子概念,表示为变量$B_1, B_2, \cdots, B_m$,*roleSet*集合中存放的是原子角色,表示为变量$r_1, r_1, \cdots, r_n$.

4.1.1 可满足概念构造算法

可满足概念生成器在生成可满足概念子术语集的过程中,需要借助概念之间的依赖关系,首先生成语义依赖度为1的简单概念;然后在简单概念的基础之上,遵循语义依赖度的分布原则,在语义簇内部的各概念之间,语义依赖度逐渐递增,在各个语义簇之间,保持其语义依赖度不变;通过前两个阶段生成的可满足概念集合满足了最大语义依赖度和语义簇数目的要求,最后再生成剩下的可满足概念,确保概念总数符合给定的参数.

算法1. 可满足概念构造算法.

输入:*satnum*:可满足概念数目

λ:最大语义依赖度

cluster:语义簇数目

conTab:构造算子表Constructor table

operTab:操作数表Operand table

输出:*S*:可满足概念集

```
1.  S←atomSet, k=1
2.  WHILE (cluster>0)
3.    d=0
4.    constructor_k←Select(seTab,asTab)
5.    operand_k←Select(atomSet,roleSet)
6.    S_k←makeAxiom(constructor_k,operand_k)
7.    S.add(S_k)
8.    d++,k++
9.    WHILE (d≤λ)
10.     constructor_k←Select(seTab,iuTab,asTab)
11.     operand_k←Select(atomSet,roleSet)∪{S_{k-1}}
12.     S_k←makeAxiom(constructor_k,operand_k)
13.     S.add(S_k)
14.     d++,k++
15.   cluster--
16. num=satnum-size(S)
17. WHILE (num>0)
18.   constructor_k←Select(seTab,asTab)
19.   operand_k←Select(atomSet,roleSet)
20.   S_k←makeAxiom(constructor_k,operand_k)
21.   S.add(S_k)
22.   k++,num--
23. RETURN S
```

通过具体的例子来解释算法1的执行过程,假设操作数表里用作种子的原子概念和原子角色个数*m*和*n*均为10,构造参数*satnum*为60,最大语义依赖度λ为4,语义簇数目*cluster*为10,生成器在以上构造参数指导下,分3步完成可满足概念子术语集的生成:

步骤1.由于原子概念本身就是可满足的,因此首先将用作种子的原子概念置于可满足概念集里,并用*k*来标记新生成的可满足概念(第1行).接下来生成语义依赖度为1的简单概念,并用*d*来标记最大语义依赖度,在此过程中,分别从*seTab*子表和*asTab*子表里取出两个构造算子放入第*k*个算子数组$constructor_k$里(第4行),这里假设从*seTab*子表里随机取出的是"≡",从*asTab*子表里随机取出的是"∃".再从*atomSet*集合和*roleSet*集合里分别取出一个原子概念和一个原子角色放入第*k*个操作数数组$operand_k$里(第5行),这里假设取出的原子概念是B_1,原子角色是r_1.然后将$constructor_k$里的两个算子"≡"和"∃"以及$operand_k$里的两个操作数B_1和r_1构造成一个可满足公理$S_1 \equiv \exists r_1.B_1$(第6行),从而生成可满足概念$S_1$并将其存放于可满足概念集*S*中(第7行).最后,*k*和*d*均递增1(第8行).

步骤2. 在简单概念 S_1 的基础之上，遵循语义依赖度的分布原则，生成一系列复杂概念，在此过程中，分别从 *seTab*、*iuTab* 和 *asTab* 这 3 个子表里，每个子表随机取出一个构造算子放入第 k 个算子数组 $constructor_k$ 里(第 10 行)，这里假设从 *seTab* 子表里取出的是“$\sqsubseteq$”，从 *iuTab* 子表里取出的是“$\sqcup$”，从 *asTab* 子表里随机取出的是“$\forall$”. 再从 *atomSet* 集合和 *roleSet* 集合里分别取出一个原子概念和一个原子角色，连同第 6 行中生成的 S_{k-1} 一起放入第 k 个操作数数组 $operand_k$ 里(第 11 行)，这里假设取出的原子概念是 B_2，原子角色是 r_2，此时 S_{k-1} 是 S_1. 然后将 $constructor_k$ 里的 3 个算子“$\sqsubseteq$”、“$\sqcup$”和“$\forall$”，以及 $operand_k$ 里的 3 个操作数 B_2、r_2 和 S_1 构造成一个可满足公理 $S_2 \sqsubseteq \forall r_2.S_1 \sqcup B_2$(第 12 行)，从而生成可满足概念 S_2 并将其添加进可满足概念集 S 中(第 13 行). 最后，k 和 d 再递增 1(第 14 行). 此时可知 S_2 的语义依赖度为 2，重复步骤 2 的操作直到生成的概念达到最大语义依赖度的要求. 最后所得到的公理如下

$S_1 \equiv \exists r_1.B_1$，$S_2 \sqsubseteq \forall r_2.S_1 \sqcup B_2$，$S_3 \sqsubseteq \exists r_3.S_2 \sqcap B_3$，$S_4 \sqsubseteq \forall r_4.S_3 \sqcap B_4$.

将标记语义簇数目的 *cluster* 递减 1(第 15 行). 此时第 1 个语义簇的生成工作结束了，该语义簇包括 4 个概念，语义依赖度分别为 1，2，3，4，满足最大语义依赖度为 4 的要求. 接下来，重复步骤 1 至步骤 2 的工作，直至生成全部的语义簇.

步骤3. 业已生成的 10 个语义簇包括 40 个概念，除去 10 个原子概念，剩余待生成的概念为 10 个，按照与步骤 1 相同的方法生成 10 个简单概念(第 18～21 行). 则所有的 60 个可满足概念均已生成.

算法 1 的执行时间主要受语义簇数目 *cluster* 和语义簇内部的最大语义依赖度 λ 所影响，因此，时间复杂度为 $o(cluster \times \lambda)$.

4.1.2　可满足概念构造算法正确性证明

可满足概念构造算法根据设定的构造参数构造出术语集的全部可满足概念，算法正确性证明如下.

证明.　由语义依赖关系构造出的术语集 $\mathcal{T}$，存在着众多的语义簇，根据定义 7，语义簇是由依赖关系连接的概念所构成的子术语集 $\mathcal{T}' \subseteq \mathcal{T}$. 算法 1 由构造参数 *cluster* 控制语义簇数目，第 1 个 WHILE 循环(第 2～15 行)每执行一次，*cluster* 减去 1，直至为 0. 同时，在语义簇内部，根据式(3)～(6)的依赖分布，在最大语义依赖度 λ 的控制下，逐次生成依赖度递增的复杂概念，算法 1 由 d 标记最大语义依赖度，第 2 个 WHILE 循环(第 9～14 行)每执行一次，d 加 1，直至为 λ. 因此，算法 1 构造出的语义簇的数目 *cluster* 和语义簇内部的最大语义依赖度 λ 符合给定构造参数的要求.

接下来，算法 1 统计已生成的可满足概念数目(第 16 行)而得到剩余待生成的概念数目 *num*，由于这些概念彼此之间不存在依赖关系，算法 1 的第 3 个 WHILE 循环仅仅生成 *num* 数目的简单概念. 因此，算法 1 构造出的可满足概念数目也符合给定构造参数的要求.　　证毕.

4.2　不可满足概念生成器

不可满足概念的构造部件也包括构造算子和操作数两部分(如表 2 所列). 在不可满足概念生成器中，构造算子特指 $\sqcap$、$\sqcup$、$\exists$、$\forall$、$\sqsubseteq$、$\equiv$ 和 $\neg$ 这 7 种，操作数同样指原子概念和原子角色两种.

表 2　不可满足概念生成器构造部件

Constructor table		Syntax
sedTab	*subClassOf*	$C_1 \sqsubseteq C_2$
	equivalentClass	$C_1 \equiv C_2$
	disjointWith	$C_1 \sqsubseteq \neg C_2$
asTab	*allValuesFrom*	$\forall t_1.A_1$
	someValuesFrom	$\exists t_1.A_1$
cTab	*complementOf*	$\neg A_1$
iTab	*intersectionOf*	$C_1 \sqcap C_2$
Operand table		
atom.sat	$B_1, B_2, \cdots, B_m$	
role.sat	$r_1, r_1, \cdots, r_n$	
atom.unsat	$A_1, A_2, \cdots, A_p$	
role.unsat	$t_1, t_1, \cdots, t_q$	

根据构造算子的种类不同，构造算子表 Constructor table 可划分为 4 个子表，*sedTab* 子表存放包含关系($\sqsubseteq$)、等价关系($\equiv$)和不相交关系($\sqsubseteq\neg$) 3 种算子，*asTab* 子表存放全称量词($\forall$)和存在量词($\exists$)两种算子，*cTab* 子表存放否定($\neg$)算子，*cTab* 子表存放合取($\sqcap$)算子.

概念之所以不可满足是因为在概念的定义过程中存在着逻辑矛盾，也就是说，存在 $C \equiv A \sqcap \neg A$ 或 $C \sqsubseteq A \sqcap \neg A$ 这种形式的逻辑矛盾. 因此将操作数里的原子及其相关的角色分为两类，一类专门提供生成不可满足概念的原子(*atom.unsat*)和角色(*role.unsat*)；另一类作为不可满足概念的附加成分，它对不可满足性不产生影响，表示为 *atom.sat* 和 *role.sat*，它们在公理中以类似于 $C \equiv A \sqcap \neg A \sqcap B$ 或 $C \sqsubseteq A \sqcap \neg A \sqcap B$ 这种形式出现，这里的 A 来自于 *atom.unsat*，B 则来自于 *atom.sat*，两者交集为空. 因此表 2 中的操作数表 Operand table 可划分为 4 类操作数集合，前

两类应用于可满足概念，*atom.sat* 集合中存放的是 $B_1,B_2,\cdots,B_m$，*role.sat* 集合中存放的是 $r_1,r_1,\cdots,r_n$. 后两类应用于不可满足概念，*atom.unsat* 集合中存放的是 $A_1,A_2,\cdots,A_m$，*role.unsat* 集合中存放的是 $t_1,\cdots,t_n$.

4.2.1 不可满足概念构造算法

Kalyanpur[24] 指出，不一致术语集中的不可满足概念并非独立存在的，它极有可能是其他不可满足概念的子概念或者父概念，为了探查不可满足概念之间的依赖关系，它将不可满足概念划分为两类：根类和派生类.

定义 10. 根不可满足概念. 如果某个概念的定义本身存在逻辑矛盾，则该概念为根不可满足概念. 它进一步可划分为纯根不可满足概念和相互依赖根不可满足概念.

例如，$\alpha_1:C_p\equiv A_1\sqcap\neg A_1$，其中 A_1 是一个原子概念，则 C_p 为纯根不可满足概念.

又如，$\alpha_2:C_{m1}\sqsubseteq\neg C_{m2}$，$\alpha_3:C_{m2}\equiv C_{m1}$，则 C_{m1} 和 C_{m2} 为相互依赖根不可满足概念.

定义 11. 派生不可满足概念. 如果出现在某个概念的逻辑矛盾直接或间接地由其他不可满足概念所造成的，则该概念为派生不可满足概念.

例如，$\alpha_4:C_1\sqsubseteq C_p$，$\alpha_5:C_2\equiv C_{m2}$，$\alpha_6:C_3\equiv C_1\sqcap C_2$，则 C_1、C_2、C_3 均为派生不可满足概念.

假如术语集 $\mathcal{T}$ 由上述 6 个公理组成，即 $\mathcal{T}=\{\alpha_1,\alpha_2,\alpha_3,\alpha_4,\alpha_5,\alpha_6\}$. 则 $\mathcal{T}$ 中不可满足概念的 Mups 如下

$\mathrm{Mups}(C_p)=\{\{\alpha_1\}\}$,

$\mathrm{Mups}(C_{m1})=\mathrm{Mups}\ (C_{m1})=\{\{\alpha_2,\alpha_3\}\}$,

$\mathrm{Mups}(C_1)=\{\{\alpha_1,\alpha_4\}\}$,

$\mathrm{Mups}(C_2)=\{\{\alpha_2,\alpha_3,\alpha_5\}\}$,

$\mathrm{Mups}(C_3)=\{\{\alpha_1,\alpha_4,\alpha_6\},\{\alpha_2,\alpha_3,\alpha_5,\alpha_6\}\}$.

借鉴 Kalyanpur 的思想和 Schlobach 等人的方法，在设计不可满足概念生成器时，采用基于根与派生不可满足概念的依赖标签方法生成不可满足概念及其 Mups.

定义 12. 依赖标签. 不可满足概念 C_k 的依赖标签形如 $\mathcal{L}(C_k)=\{(C_k,Index_k,Ax_k)\}\cup\mathcal{L}(C_{k-1})$，其中 $Index_k$ 是与 C_k 相关的公理索引，Ax_k 是直接生成 C_k 的公理.

一般地，依赖标签初始化为 $\mathcal{L}(C_0)=\{([A,\neg A],\varnothing,\mathrm{Null})\}$ 或者 $\mathcal{L}(C_0)=\{([C_{m1},C_{m2}],\varnothing,\mathrm{Null})\}$.

下面是上述 6 个不可满足概念的依赖标签

$\mathcal{L}(C_p)=\{([A,\neg A],\varnothing,\mathrm{Null}),$
$(C_p,\{\alpha_1\},C_p\equiv A_1\sqcap\neg A_1)\}$,

$\mathcal{L}(C_{m1})=\{([C_{m1},C_{m2}],\varnothing,\mathrm{Null}),$
$(C_{m1},\{\alpha_2,\alpha_3\},\{C_{m1}\sqsubseteq\neg C_{m2},C_{m2}\equiv C_{m1}\})\}$,

$\mathcal{L}(C_{m2})=\{([C_{m1},C_{m2}],\varnothing,\mathrm{Null}),$
$(C_{m2},\{\alpha_2,\alpha_3\},\{C_{m1}\sqsubseteq\neg C_{m2},C_{m2}\equiv C_{m1}\})\}$,

$\mathcal{L}(C_1)=\{([A,\neg A],\varnothing,\mathrm{Null}),$
$(C_p,\{\alpha_1\},C_p\equiv A_1\sqcap\neg A_1),$
$(C_1,\{\alpha_1,\alpha_4\},C_1\sqsubseteq C_p)\}$,

$\mathcal{L}(C_2)=\{([C_{m1},C_{m2}],\varnothing,\mathrm{Null}),$
$(C_{m1},\{\alpha_2,\alpha_3\},\{C_{m1}\sqsubseteq\neg C_{m2},C_{m2}\equiv C_{m1}\}),$
$(C_2,\{\alpha_2,\alpha_3,\alpha_5\},C_2\equiv C_{m2})\}$,

$\mathcal{L}(C_3)=\{(\{[A,\neg A],[C_{m1},C_{m2}]\},\varnothing,\mathrm{Null}),$
$(C_p,\{\alpha_1\},C_p\equiv A_1\sqcap\neg A_1),$
$(C_{m2},\{\alpha_2,\alpha_3\},\{C_{m1}\sqsubseteq\neg C_{m2},C_{m2}\equiv C_{m1}\}),$
$(C_1,\{\alpha_1,\alpha_4\},C_1\sqsubseteq C_p),$
$(C_2,\{\alpha_2,\alpha_3,\alpha_5\},C_2\equiv C_{m2}),$
$(C_3,\{\{\alpha_1,\alpha_4,\alpha_6\},\{\alpha_2,\alpha_3,\alpha_5,\alpha_6\}\},$
$C_3\equiv C_1\sqcap C_2)\}$.

基于依赖标签的不可满足概念生成器算法如下.

算法 2. 不可满足概念构造算法.

输入：*unsatnum*：不可满足概念的数目
$\mathcal{N}_{\max}$：冲突公理集最大基数
$\mathcal{L}_{\max}$：冲突公理最大基数
conTab：构造算子表 Constructor table
operTab：操作数表 Operand table

输出：*U*：不可满足概念集
Mups($\mathcal{T}$)：术语集的所有 Mups

1. $n=1,i=1,j=0,k=1$
2. WHILE($n\leq\mathcal{N}_{\max}$)
3. $constructor_n=\{equivalentClass,intersectionOf\}$
4. $constructor_n=constructor_n\cup cTab\cup Select^*(asTab)$
5. $operand_n\leftarrow Select(atom.unsat,role.unsat)$
6. $C_{pn}=makeAxiom(constructor_n,operand_n)$
7. $U_p=U_p\cup\{C_{pn}\},n{+}{+}$
8. $\mathrm{Mups}(C_{pn})=\{\{\alpha_{pn}\}\}$,Mups($\mathcal{T}$).add(Mups($C_{pn}$))
9. WHILE ($i\leq\mathcal{N}_{\max}$)
10. $constructor_k=\{subClassOf,intersectionOf\}$
11. $constructor_k=constructor_k\cup Select^*(asTab)$
12. $C_{pi}\leftarrow Select(U_p)$
13. $operand_k\leftarrow Select(atom.sat,role.sat)$
14. IF ($i==1$) $operand_k=\{C_{pi}\}\cup operand_k$
15. ELSE $operand_k=\{C_{k-1},C_{pi}\}\cup operand_k$

16. $C_k = makeAxiom(constructor_k, operand_k)$
17. $U = U \cup \{C_k\}, i{+}{+}, k{+}{+}$
18. $\mathrm{Mups}(C_k) = \mathrm{Mups}(C_{k-1}).\mathrm{add}(\alpha_k)$
19. $\mathrm{Mups}(\mathcal{T}).\mathrm{add}(\mathrm{Mups}(C_k))$
20. $constructor_j = \{subClassOf, equivalentClass\} \cup cTab$
21. $operand_j \leftarrow \{C_{mk1}, C_{mk2}\}$
22. $\{C_{mk1}, C_{mk2}\} = makeAxiom(constructor_j, operand_j)$
23. $U_m = U_m \cup \{C_{mk1}, C_{mk2}\}, j = j+2$
24. $\mathrm{Mups}(C_{mk1}) = \mathrm{Mups}(C_{mk2}) = \{\{\alpha_{mk1}, \alpha_{mk2}\}\}$
25. $\mathrm{Mups}(\mathcal{T}).\mathrm{add}\{\mathrm{Mups}(C_{mk1}), \mathrm{Mups}(C_{mk2})\}$
26. WHILE $(j \leq \mathcal{L}_{\max})$
27. $constructor_k = \{subClassOf, intersectionOf\}$
28. $constructor_k = constructor_k \cup Select(asTab)$
29. $C_{mk} \leftarrow Select(U_m)$
30. $operand_k \leftarrow Select(atom.sat, role.sat)$
31. IF $(j == 2)$ $operand_k = \{C_{mk}\} \cup operand_k$
32. ELSE $operand_k = \{C_{k-1}, C_{mk}\} \cup operand_k$
33. $C_k = makeAxiom(constructor_k, operand_k)$
34. $U = U \cup \{C_k\}, j{+}{+}, k{+}{+}$
35. $\mathrm{Mups}(C_k) = \mathrm{Mups}(C_{k-1}).\mathrm{add}(\alpha_k)$
36. $\mathrm{Mups}(\mathcal{T}).\mathrm{add}(\mathrm{Mups}(C_k))$
37. $num = unsatnum - size(U)$
38. WHILE $(num > 0)$
39. $constructor_k = \{equivalentClass, intersectionOf\}$
40. $constructor_k = constructor_k \cup cTab \cup Select^*(asTab)$
41. $operand_k \leftarrow Select(atom.unsat, role.unsat)$
42. $C_k = makeAxiom(constructor_k, operand_k)$
43. $\mathrm{Mups}(C_k) = \{\{\alpha_k\}\}, \mathrm{Mups}(\mathcal{T}).\mathrm{add}(\mathrm{Mups}(C_k))$
44. $U = U \cup \{C_k\}, k{+}{+}, num{-}{-}$
45. $U = U_p \cup U_m$
46. RETURN U, $\mathrm{Mups}(\mathcal{T})$

同样通过例子来说明算法 2 的执行过程，假设构造参数 *unsatnum* 为 30，最大冲突公理集基数 $\mathcal{N}_{\max}$ 为 4，最大冲突公理基数 $\mathcal{L}_{\max}$ 为 6. 生成器在以上构造参数指导下，分 3 步完成不可满足概念的生成：

步骤 1. 生成满足 $\mathcal{N}_{\max}$ 为 4 的不可满足概念. 首先分别从 *sedTab* 子表和 *iTab* 子表中取出等价关系（≡）和合取（⊓）两个构造算子放入 $constructor_n$ 里（第 3 行），再继续从 *cTab* 子表中取出“¬”算子，并从 *asTab* 子表中取出一对“∀、∃”构造算子，一起放入 $constructor_n$ 里（第 4 行）. 再从 *atom.unsat* 集合和 *role.unsat* 集合里分别取出一个原子概念和一个原子角色放入第 k 个操作数数组 $operand_n$ 里（第 5 行），这里假设取出的原子概念是 A_1，原子角色是 t_1. 然后将 $constructor_n$ 里的 5 个算子“≡”、“⊓”、“¬”和“∀、∃”，以及 $operand_k$ 里的两个操作数 A_1 和 t_1 构造成一个纯根不可满足公理 $\alpha_{p1}: C_{p1} \equiv \forall t_1.A_1 \sqcap \exists t_1.\neg A_1$（第 6 行），从而生成纯根不可满足概念 C_{p1} 并将其存放于纯根不可满足概念集 U_p 中，用于标记纯根不可满足概念的 n 递增 1（第 7 行），同时得到它的 $\mathrm{Mups}(C_{p1}) = \{\{\alpha_{p1}\}\}$（Line 8）. 以同样的方式生成 $\mathcal{N}_{\max} = 4$ 个纯根不可满足概念

$$\alpha_{p1}: C_{p1} \equiv \forall t_1.A_1 \sqcap \exists t_1.\neg A_1,$$
$$\alpha_{p2}: C_{p2} \equiv \forall t_2.A_2 \sqcap \exists t_2.\neg A_2,$$
$$\alpha_{p3}: C_{p3} \equiv \forall t_3.A_3 \sqcap \exists t_3.\neg A_3,$$
$$\alpha_{p4}: C_{p4} \equiv \forall t_4.A_4 \sqcap \exists t_4.\neg A_4.$$

在纯根不可满足概念的基础之上，生成一系列派生概念，首先分别从 *sedTab* 子表和 *iTab* 子表中取出包含关系（⊑）和合取（⊓）两个构造算子放入 $constructor_k$ 里（第 10 行），并从 *asTab* 子表中随机取出一个构造算子，一起放入 $constructor_n$ 里（第 11 行），这里假设取出的构造算子是“∀”. 再从 U_p 中随机取出一个纯根不可满足概念 C_{pi}（第 12 行）. 然后从 *atomSet* 集合和 *roleSet* 集合里分别取出一个原子概念和一个原子角色放入第 k 个操作数数组 $operand_k$ 里（第 13 行），这里假设取出的原子概念是 B_1，原子角色是 r_1. 如果此时待生成的是第 1 个派生不可满足概念，则将已选出的纯根不可满足概念 C_{pi} 置入操作数数组 $operand_k$ 里（第 14 行），否则将已选出的纯根不可满足概念 C_{pi} 连同前一次生成的派生不可满足概念 C_{k-1} 一起置入 $operand_k$ 里（第 15 行），最后将取出的构造算子和操作数构造成公理 $\alpha_1: C_1 \sqsubseteq C_{p1} \sqcap \forall r_1.B_1$（第 16 行），则 C_1 为派生不可满足概念，将其添加进不可满足概念集 U 中（第 17 行），同时获得它的 $\mathrm{Mups}(C_1) = \{\{\alpha_1, \alpha_{p1}\}\}$ 并添加到 $\mathrm{Mups}(\mathcal{T})$ 中（第 17～18 行）. 按照同样的方法，依次得到

$$\alpha_1: C_1 \sqsubseteq C_{p1} \sqcap \forall r_1.B_1,$$
$$\alpha_2: C_2 \sqsubseteq C_1 \sqcap C_{p2} \sqcap \forall r_2.B_2,$$
$$\alpha_3: C_3 \sqsubseteq C_2 \sqcap C_{p3} \sqcap \forall r_3.B_3,$$
$$\alpha_4: C_4 \sqsubseteq C_3 \sqcap C_{p4} \sqcap \forall r_4.B_4$$

以及它们的 Mups

$$\mathrm{Mups}(C_2) = \{\{\alpha_2, \alpha_1, \alpha_{p1}\}, \{\alpha_2, \alpha_{p2}\}\},$$
$$\mathrm{Mups}(C_3) = \{\{\alpha_3, \alpha_2, \alpha_1, \alpha_{p1}\}, \{\alpha_3, \alpha_2, \alpha_{p2}\}\{\alpha_3, \alpha_{p3}\}\},$$
$$\mathrm{Mups}(C_4) = \{\{\alpha_4, \alpha_3, \alpha_2, \alpha_1, \alpha_{p1}\}, \{\alpha_4, \alpha_3, \alpha_2, \alpha_{p2}\}, \{\alpha_4, \alpha_3, \alpha_{p3}\}, \{\alpha_4, \alpha_{p4}\}\}.$$

此时，生成的不可满足概念 C_4 的冲突公理集基数为 4. 符合冲突公理集最大基数为 4 的要求.

步骤2. 生成满足$\mathcal{L}_{\max}$为6的不可满足概念. 首先分别从 *sedTab* 子表中取出包含关系($\sqsubseteq$)和等价关系($\equiv$)两个构造算子,再从 *cTab* 子表中取出"$\neg$"算子一起放入 $constructor_j$ 里(第20行),并将 C_{mk1} 与 C_{mk2} 作为操作数,生成一对相互依赖根不可满足概念 $\alpha_{mk1}: C_{mk1} \sqsubseteq \neg C_{mk2}$, $\alpha_{mk2}: C_{mk1} \equiv C_{mk2}$(第21~22行),将其存入相互依赖根不可满足概念集 U_m 并获取其 Mups 为 $\text{Mups}(C_{mk1}) = \text{Mups}(C_{mk2}) = \{\{\alpha_{mk1}, \alpha_{mk2}\}\}$(第23~24行).

在相互依赖根不可满足概念的基础之上,生成一系列派生不可满足概念. 首先分别从 *sedTab* 子表和 *iTab* 子表中取出包含关系($\sqsubseteq$)和合取($\sqcap$)两个构造算子放入 $constructor_r$ 里(第27行),并从 *asTab* 子表中随机取出一个构造算子,一起放入 $constructor_n$ 里(第28行),这里假设取出的构造算子是"$\forall$". 再从 U_m 中随机取出一个相互依赖根不可满足概念 C_{mk1}(第29行). 然后从 *atomSet* 集合和 *roleSet* 集合里分别取出一个原子概念和一个原子角色放入第 k 个操作数数组 $operand_k$ 里(第30行),这里假设取出的原子概念是 B_5,原子角色是 r_5. 如果此时待生成的是第1个基于相互依赖根的派生不可满足概念,则将已选出的相互依赖根不可满足概念 C_{mk} 置入操作数数组 $operand_k$ 里(第31行),否则将已选出的 C_{mk} 连同前一次生成的派生不可满足概念 C_{k-1} 一起置入 $operand_k$ 里(第32行). 最后将取出的构造算子和操作数构造成公理 $\alpha_5: C_5 \sqsubseteq C_{mk1} \sqcap \forall r_5.B_5$(第33行),则 C_5 为派生不可满足概念,将其添加进不可满足概念集 U 中(第34行),同时获得它的 $\text{Mups}(C_5) = \{\{\alpha_5, \alpha_{mk1}, \alpha_{mk2}\}\}$(第35行). 按照同样的方法,依次得到

$$\alpha_6: C_6 \sqsubseteq C_5 \sqcap \forall r_6.B_6,$$
$$\alpha_7: C_7 \sqsubseteq C_6 \sqcap \forall r_7.B_7,$$
$$\alpha_8: C_8 \sqsubseteq C_7 \sqcap \forall r_8.B_8$$

以及它们的 Mups

$$\text{Mups}(C_6) = \{\{\alpha_6, \alpha_5, \alpha_{mk1}, \alpha_{mk2}\}\},$$
$$\text{Mups}(C_7) = \{\{\alpha_7, \alpha_6, \alpha_5, \alpha_{mk1}, \alpha_{mk2}\}\},$$
$$\text{Mups}(C_8) = \{\{\alpha_8, \alpha_7, \alpha_6, \alpha_5, \alpha_{mk1}, \alpha_{mk2}\}\}.$$

此时,生成的不可满足概念 C_8 的冲突公理基数为6. 符合冲突公理最大基数为6的要求.

步骤3. 目前已经生成了16个不可满足概念,剩余待生成的不可满足概念为14个(第37行),按照与步骤1相同的方法生成14个根不可满足概念(第40~46行). 所有的30个不可满足概念均已生成.

算法2的执行时间同时受不可满足概念数目 *unsatnum*、冲突公理集最大基数 $\mathcal{N}_{\max}$ 以及冲突公理最大基数 $\mathcal{L}_{\max}$ 所影响,时间复杂度为线性时间阶 $o(n)$.

4.2.2 不可满足概念构造算法正确性证明

采用基于根与派生不可满足概念的依赖标签方法设计出的不可满足概念构造算法,着眼于 Mups 的复杂程度,通过冲突公理集最大基数和冲突公理最大基数这两个构造参数,构造出全部的不可满足概念及其 Mups,因此,算法的正确性证明包括两部分,一部分是 Mups 的正确性,另一部分是构造出的术语集符合 *unsatnum*、$\mathcal{N}_{\max}$ 以及 $\mathcal{L}_{\max}$ 的参数设定.

首先 Mups 的正确性,Schlobach 等人[23]使用基于标签公式(*labeled formula*)的 Tableau 演算扩展规则来计算不可满足概念的 Mups. 标签公式形如 $(a:C)^x$, a 是 Tableau 演算过程中产生的个体断言,C 是概念,x 是一组公理.

定理1. 设 C_k 是关于术语集 $\mathcal{T}$ 中的不可满足概念,依赖标签 $\mathcal{L}(C_k)$ 的公理索引集合 $Index_k$ 则为 C_k 的 Mups.

证明. 根据 Mups 的定义(定义2),首先证明由 $Index_k$ 的公理所构成的术语集 $\mathcal{T}'$ 关于 C_k 是不可满足的. 根据基于标签公式的 Tableau 演算扩展规则,Tableau 先从一个单一的分支 $B := B \cup \{(a:C_k)^{Index_k}\}$ 开始扩展,C_k 是由 $\alpha_k: C_k \sqsubseteq C_{k-1} \sqcap S_k$ 所生成的,其中 S_k 是可满足概念,对 C_k 的不可满足性无影响,因此,Tableau 分支扩展成 $B := B \cup \{(a:C_{k-1})^{Index_k}\}$,其中 $Index_k = \{\alpha_k\} \cup Index_{k-1}$. 类似地,$C_{k-1}$ 由 $\alpha_{k-1}: C_{k-1} \sqsubseteq C_{k-2} \sqcap S_{k-1}$ 生成,Tableau 分支进一步扩展为 $B := B \cup \{(a:C_{k-2})^{Index_{k-1}}\}$,其中 $Index_{k-1} = \{\alpha_{k-1}\} \cup Index_{k-2}$. 以此类推,$C_1$ 由 $\alpha_1: C_1 \sqsubseteq C_0 \sqcap S_1$ 生成,此时的 Tableau 分支扩展为 $B := B \cup \{(a:C_0)^{Index_1}\}$,其中 $Index_1 = \{\alpha_1\} \cup Index_0$. 因为生成 C_0 的公理或者形如 $\alpha_{p1}: C_{p1} \equiv \forall t_1.A_1 \sqcap \exists t_1.\neg A_1$,或者形如 $\alpha_{mk1}: C_{mk1} \sqsubseteq \neg C_{mk2}$, $\alpha_{mk2}: C_{mk1} \equiv C_{mk2}$,所以 Tableau 分支最终扩展 $B := B \cup \{(a:A)^{Index_1}, (a:\neg A)^{Index_1}\}$ 或者 $B := B \cup \{(a:C_{mk1})^{Index_1}, (a:C_{mk2})^{Index_1}\}$,其中 $Index_1$ 形如 $\{\alpha_{p1}\}$ 或 $\{\alpha_{mk1}, \alpha_{mk2}\}$. 由于在分支上出现了一对相互矛盾的概念,所以 B 封闭. 因此,C_k 在术语集 $\mathcal{T}'$ 中是不可满足的.

设 $\mathcal{T}''$ 是从 $Index_k$ 里删除任意一个公理 α_i 之后剩余的公理所构成的子术语集,仍然采用基于标签公式的 Tableau 演算扩展规则方法进行证明. 由于术语集 $\mathcal{T}'$ 中的公理形如 $\alpha_k: C_k \sqsubseteq C_{k-1}$, …,

$\alpha_i:C_i\sqsubseteq C_{i-1},\cdots,\alpha_1:C_1\sqsubseteq C_0$，从 $Index_k$ 中删除 α_i 相当于从 $\mathcal{T}'$ 中删除 $\alpha_i:C_i\sqsubseteq C_{i-1}$ 以及从 Tableau 中剪去分支 $\{(a:C_{i-1})^{Index_1}\}$，因为 C_i 是 C_0 的子集，C_0 的公理或者形如 $\alpha_{p1}:C_{p1}\equiv\forall t_1.A_1\sqcap\exists t_1.\neg A_1$，或者形如 $\alpha_{mk1}:C_{mk1}\sqsubseteq\neg C_{mk2},\alpha_{mk2}:C_{mk1}\equiv C_{mk2}$. 这意味着分支 B 中不会同时存在 $(a:A)^{Index_1}$ 和 $(a:\neg A)^{Index_1}$，也不会同时存在 $(a:C_{mk1})^{Index_1}$ 和 $(a:C_{mk2})^{Index_1}$. 所以，C_k 在术语集 $\mathcal{T}''$ 中是可满足的.　　证毕.

然后证明构造出的术语集符合不可满足概念数目 $unsatnum$、冲突公理集最大基数 $\mathcal{N}_{\max}$ 以及冲突公理最大基数 $\mathcal{L}_{\max}$ 的参数设定，证明如下.

证明.　根据定义 10，算法 2 的第 1 个 WHILE 循环(第 2～8 行)首先生成 $\mathcal{N}_{\max}$ 个纯根不可满足概念及其 Mups，接下来在每一个纯根不可满足概念的基础之上，根据定义 11，算法 2 的第 2 个 WHILE 循环(第 9～19 行)依次生成对应的派生不可满足概念及其 Mups，在此过程中，冲突公理集最大基数 $\mathcal{N}_{\max}$ 保持不变. 因此符合 $\mathcal{N}_{\max}$ 的参数设定.

同样根据定义 10 与定义 11，生成冲突公理最大基数为 $\mathcal{L}_{\max}$ 的不可满足概念及其 Mups，算法 2 由 j 标记冲突公理最大基数，第 3 个 WHILE 循环(第 26～36 行)每执行一次，j 加 1，直至为 $\mathcal{L}_{\max}$. 因此，算法 2 构造出的子术语集(对应不可满足概念部分)符合给定构造参数 $\mathcal{N}_{\max}$ 和 $\mathcal{L}_{\max}$ 的要求.

接下来，统计已生成的不可满足概念数目(第 37 行)而得到剩余待生成的不可满足概念数目 num，算法 2 的第 4 个 WHILE 循环便生成 num 个简单的根不可满足概念. 因此，算法 2 构造出的不可满足概念数目也符合给定构造参数的要求.　　证毕.

5　推理机性能评测

推理机性能评测实验是在 PC 机 Windows7 操作系统(Intel(R) Core(TM) CPU 3.40GHz，4.00GB RAM)下进行的. 为了评测推理机在黑盒算法下求解不一致术语集的性能，MupsBen 在不同量级的构造参数指导下生成了一系列复杂度不同的术语集测试数据，为了避免交叉影响，每次实验只针对一类构造参数，设置不同的参数值生成不同的测试数据，评测不同推理机的推理性能. 性能评测实验所用的黑盒算法来自 Kalyanpur[24] 的 MUPS-HST 算法，所用推理机的版本号依次为 Pellet-2.3.1、Hermit-1.3.8、FaCT++-1.6.2、JFact-1.0.0 和 TrOWL-1.4. MUPS-HST 算法通过调用外部推理机能够求解出不一致术语集中不可满足概念的 Mups，作者在该文献[24]中证明了在 MUPS-HST 算法的保证下，推理机求解 Mups 的结果都是正确的，这一特点是由算法本身所决定的. 为了便于说明，设不可满足概念数目为 β，语义簇数目为 k，最大语义依赖度为 λ，最大冲突公理集基数为 n，最大冲突公理基数为 l.

由于评测实验的目标是检验所定义的复杂度指标(表现为构造参数)的适用性，即该指标能否有效反映 Mups 求解问题的数据复杂程度，在此基础之上评测不同推理机求解 Mups 问题的性能差异. 因此，在每一组实验中都将 6 个构造参数划分为一个可变参数和 5 个不变参数，在针对某一个构造参数进行实验时，该构造参数即为可变参数，在参数合理的取值范围之内变动其参数值从而取不同的评测点，其他 5 个构造参数为不变参数，由于这些不变参数取值始终不变，因而在同一组对比实验中对推理机性能评测结果不产生影响，所以它们可以在其取值范围内取任意值. 例如在大规模 TBox 数据(可满足概念数目 α 取值为 20000)情形下，评测不可满足概念数目这个构造参数的适用性，如果作为可变的参数值，理论上不可满足概念数目 β 可以取任意值，但实际中考虑到本体中不可满足概念数目远小于可满足概念数目，因此不适合取较大的值，同时，由于不可满足概念数目的增多会导致推理机求解 Mups 问题时间的剧增，最终无法求出问题的解. 所以，从 β 的取值范围中截取一段合理的取值区间为 10～50，在该变动范围内以 10 为增幅取 5 个评测点进行评测实验. 按相同的处理办法，k 变动范围为 20～100(增幅为 20)，λ 变动范围为 2～6(增幅为 1)，n 变动范围为 1～5(增幅为 2)，l 变动范围为 4～12(增幅为 2). 对于不变参数，理论上同样可以任意取值，但具体到实际情形，在合理的取值范围之内，将 β 取值为 20，k 取值为 50，λ 取值为 5，n 取值为 2，l 取值为 5.

第 1 组实验评测聚集度取最小与最大值时 5 种推理机推理时间的差异，当聚集度取得最小值 $\eta_{\min}=0$ 时 $k=0,\lambda=1$，此时，MupsBen 生成的测试数据复杂性最低，因为可满足概念生成器生成的子术语集不存在任何语义簇，全部概念都为语义依赖度为 1 的简单概念，再设置其他构造参数 $\alpha=1000$，$\beta=50,n=5,l=5$，用黑盒算法读入生成的测试术语集，调用 5 种推理机求解 Mups，如图 2 所示，执行时间为 min 情形下所对应的纵坐标值；当聚集度取得

最大值 $\eta_{\max}=\frac{1+\lambda}{2\lambda}$(或 $\eta_{\max}=\frac{k+\alpha}{2\alpha}$)时,MupsBen 生成的测试数据复杂性最高,因为可满足概念生成器生成的子术语集不存在非语义簇的任何简单概念,设置构造参数 $\alpha=1000,\beta=50,k=100,\lambda=10,n=5,l=5$,按照同样的方式,执行时间为 max 情形下所对应的纵坐标值.

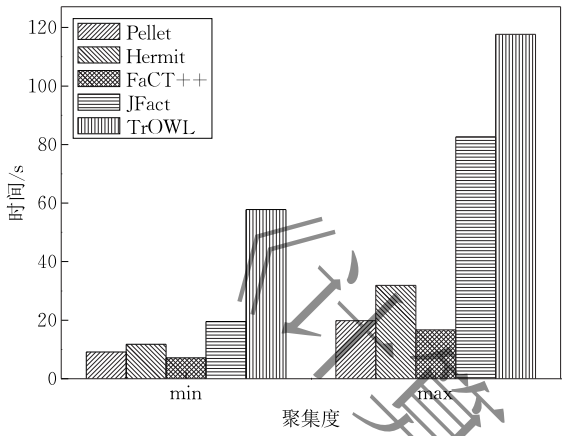


图 2 聚集度取最小与最大值时的执行时间

第 2 组实验评测可满足概念数目 α 的变化对 5 种推理机性能的影响(不变参数 $\beta=20,k=50,\lambda=5,n=2,l=5$),纵坐标取对数坐标,如图 3 所示.一方面,可满足概念越多,术语集的规模越大,黑盒算法所调用的推理机每一次对术语集进行一致性检测所花费的时间就越多,因而,推理机的性能随可满足概念数目的增大而降低.另一方面,当可满足概念数目较少时,FaCT++的性能最好,Pellet 次之,JFact 好于 TrOWL.然而,随着可满足概念数目的剧增,诸推理机的性能有了很大的变化,TrOWL 最先反超 JFact,随后 Pellet 和 Hermit 依次反超 FaCT++,其中,JFact 的性能下降最为剧烈.

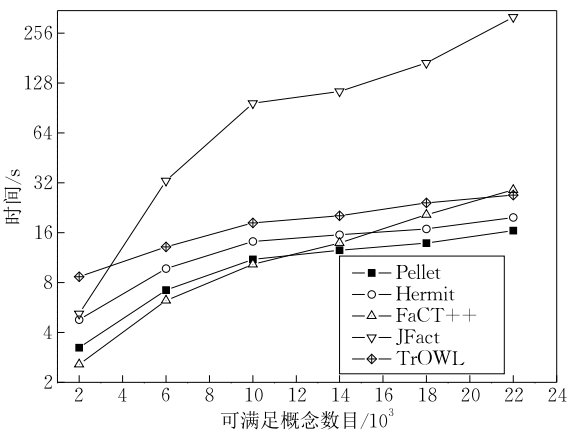


图 3 执行时间随可满足概念数目变化折线图

由于在不同量级的 TBox 规模之下,诸推理机的性能出现显著地变化,因此,接下来的评测实验分别针对较小规模和较大规模两种情形下的 TBox 测试数据进行评测.

第 3 组实验评测不可满足概念数目 β 的变化对 5 种推理机性能的影响(较小规模 TBox 下不变参数 $\alpha=1000,k=50,\lambda=5,n=5,l=5$;较大规模 TBox 下不变参数 $\alpha=20\,000,k=50,\lambda=5,n=2,l=5$),纵坐标也取对数坐标,如图 4 所示.不可满足概念数目的增加意味着不一致术语集的 Mups 数量的增多,对于每一个不可满足概念,黑盒算法都要对整个术语集执行一轮扩展和压缩,因此,推理时间自然增多.TBox 规模较小情况下,FaCT++性能最好,TrOWL 最差,TBox 规模较大情况下,不可满足概念数目小于 10 时,Pellet、Hermit 与 TrOWL 依次好于 FaCT++,当不可满足概念数目递增时,FaCT++的性能很快就达到最好.

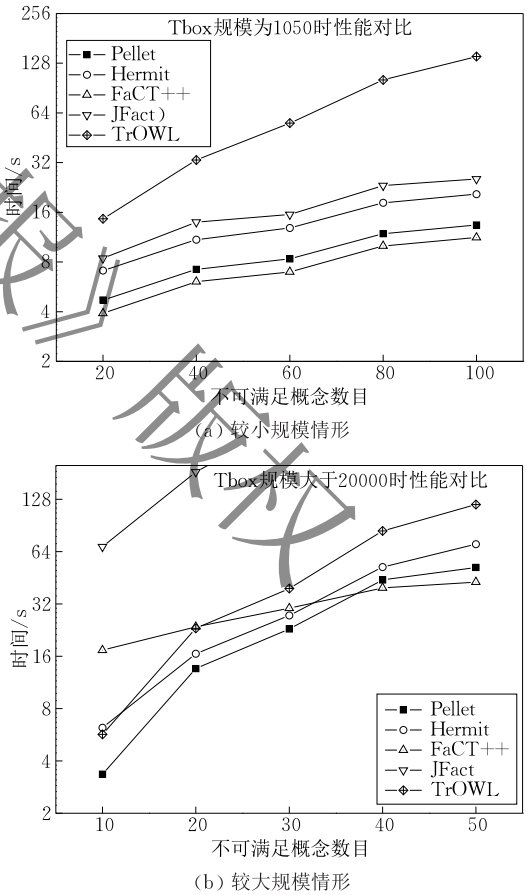
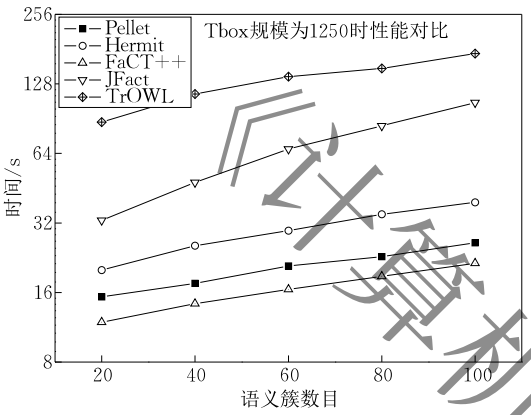


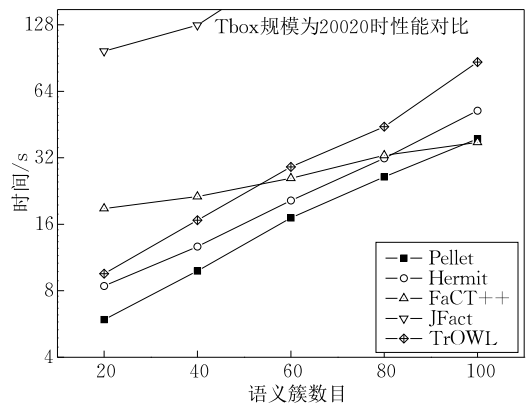
图 4 执行时间随不可满足概念数目变化折线图

第 4 组实验评测语义簇数目 k 的变化对 5 种推理机性能的影响(较小规模 TBox 下不变参数 $\alpha=1200,\beta=50,\lambda=10,n=5,l=5$;较大规模 TBox 下

不变参数 $\alpha=20\,000,\beta=20,\lambda=5n=2,l=5$），纵坐标仍取对数坐标，如图 5 所示. 由于术语集中的概念是通过依赖关系构造出来的，而语义簇所包括的概念彼此之间都存在依赖关系，语义簇数目的增加决定着依赖关系数量的增加，这种增加会导致推理机耗费更多的时间去进行可满足性检测，从而影响其推理性能. TBox 规模较小情况下的各推理机性能对比没有变化，TBox 规模较大情况下，FaCT++分别在语义簇数目为 60、80 和 100 时超过 Pellet、Hermit 和 TrOWL.



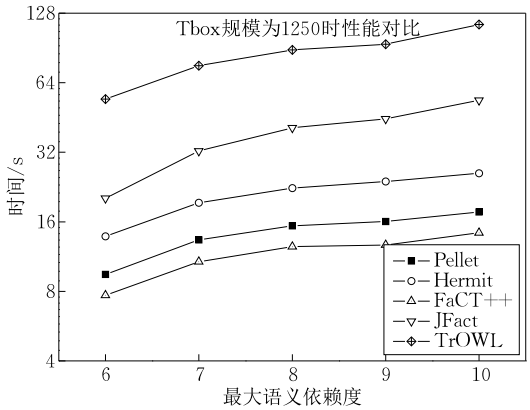
(a) 较小规模情形



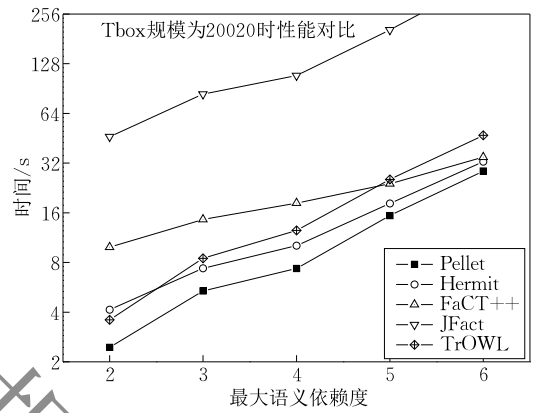
(b) 较大规模情形

图 5　执行时间随语义簇数目变化折线图

第 5 组实验评测最大语义依赖度 λ 的变化对 5 种推理机性能的影响（较小规模 TBox 下不变参数 $\alpha=1200,\beta=50,k=50,n=5,l=5$；较大规模 TBox 下不变参数 $\alpha=2000,\beta=20,k=50,n=2,l=5$），纵坐标继续取对数坐标，如图 6 所示. 最大语义依赖度指标在术语集中最直接的表现就是某一个概念与其他概念存在着较多的依赖关系，这种依赖关系使得越处于依赖关系图层次结构下层的概念越复杂，从而在推理机对这种概念进行可满足性检测时，耗费的时间越多，因而性能就会越低. TBox 规模较小情况下，FaCT++依然保持其性能优势；当 TBox 规模较大时，Pellet 在最大语义依赖度递增过程中一直具有最好的性能.



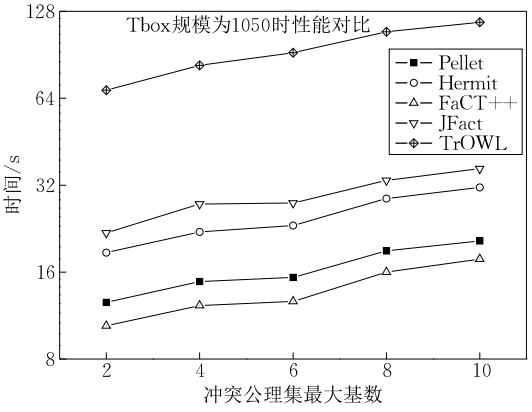
(a) 较小规模情形



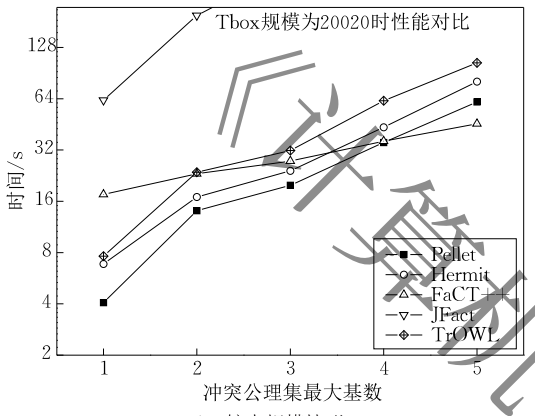
(b) 较大规模情形

图 6　执行时间随最大语义依赖度变化折线图

第 6 组实验评测冲突公理集最大基数 n 的变化对 5 种推理机性能的影响（较小规模 TBox 下不变参数 $\alpha=1000,\beta=50,k=50,\lambda=5,l=15$；较大规模 TBox 下不变参数 $\alpha=20\,000,\beta=20,k=50,\lambda=5,l=5$），纵坐标同样取对数坐标，如图 7 所示. 由于 MupsBen 在生成不一致术语集时，采用了基于根和派生依赖的方法，冲突公理集最大基数越大，造成某个不可满足概念的极小冲突公理集越多，而黑盒算法需要找到所有的这种极小冲突公理集，因此，推理机需要耗费的时间自然越多. 在 TBox 规模较小情况下，除了 JFact 在个别评测点上的性能出现波动之外，其他推理机性能对比与前面各组实验结果均相同，若从整体来看，JFact 的推理机性能仍然具有随复杂程度增加而降低的趋势. 在 TBox 规模较大的情况下，随着冲突公理集最大基数取值的增加，FaCT++的推理时间增加较慢，相比之下，Pellet、Hermit 和 TrOWL 的推理时间增加较快，因此，当冲突公理集最大基数超过 4 时，FaCT++的性能后来居上，胜过其他推理机.

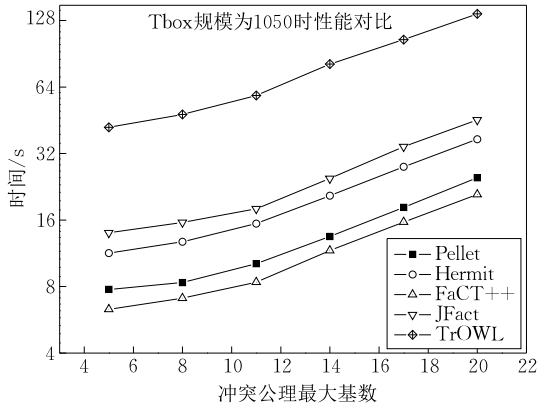


(a) 较小规模情形

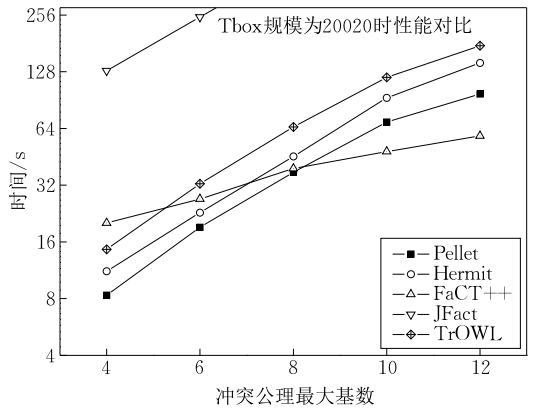


(b) 较大规模情形

图 7 执行时间随冲突公理集最大基数变化折线图



(a) 较小规模情形



(b) 较大规模情形

图 8 执行时间随冲突公理最大基数变化折线图

第 7 组实验评测冲突公理最大基数 l 的变化对 5 种推理机性能的影响(较小规模 TBox 下不变参数 $\alpha=1000,\beta=50,k=50,\lambda=5,n=5$;较大规模 TBox 下不变参数 $\alpha=1000,\beta=20,k=50,\lambda=5,n=2$),纵坐标同样也取对数坐标,如图 8 所示. 冲突公理最大基数与术语集的最大语义依赖度有密切的关系,某个不可满足概念的冲突公理最大基数越大,表明从根不可满足概念到该不可满足概念所经历的依赖关系和派生概念就越多,在推理机求解 Mups 的时候,涉及到的不可满足概念就越多,这些都影响着推理机的性能. 在 TBox 规模较小情况下推理机性能没有变化,TBox 规模较大情况下,冲突公理最大基数小于 8 时,Pellet 优于 FaCT++,反之则 FaCT++优于 Pellet.

综合考虑第 2 至第 7 组实验,TBox 规模较小时,当评测的复杂度指标取某个特殊值,不一致术语集的复杂度是确定的,5 种推理机在同样复杂程度的术语集上进行求解 Mups,它们之间的性能高低差异比较稳定,由高到低依次表现为 FaCT++、Pellet、Hermit、JFact、TrOWL. 当 TBox 规模较大时,Pellet 则表现出较好的性能,Hermit 次之,TrOWL 差于 Hermit 但优于 FaCT++,而 JFact 受规模影响很大,性能下降剧烈,并可以进一步发现,在大规模数据的背景下,当复杂度较高时,JFact 在很多评测点上无法求出问题的解. 但是,随着复杂度指标参数值的增加,FaCT++的性能下降幅度小于其他推理机,因而经常能够在指标参数达到某一个值时,反超 Pellet,达到最好的性能.

为了进一步探析数据规模较大时各个推理机性能差异的原因,第 8 组实验针对 TBox 的可满足性检测这一经典推理问题进行评测,之所以选择该问题,是因为求解 Mups 问题首先需要对 TBox 进行可满足性检测,求出所有的不可满足概念,进而计算每个不可满足概念的 Mups. 实验结果如表 3 所示(不变参数 $\beta=20,k=50,\lambda=5,n=2,l=5$).

表 3 不同 TBox 规模下推理机可满足性检测时间

(单位:s)

TBox 规模	FaCT++	Pellet	Hermit	JFact	TrOWL
2020	0.109	0.156	0.250	0.546	0.078
6020	0.905	0.249	0.671	3.822	0.140
10020	2.996	0.281	1.170	13.432	0.312
14020	6.193	0.343	1.669	28.314	0.608
18020	12.153	0.406	2.371	56.765	0.796
22020	23.650	0.609	2.855	92.543	1.232

可以发现，TBox 规模增大时，Pellet、Hermit 与 TrOWL 的可满足性检测时间的增加非常有限，但是 FaCT++的可满足性检测时间增加幅度很大，而 JFact 增加的幅度尤其剧烈. 因此较大规模 TBox 下，FaCT++和 JFact 的性能会受到很大的影响. 从推理机的推理算法和优化技术的角度，可以进一步揭示推理机性能差异的深层次原因.

Pellet 和 FaCT++基于 tableaux 算法，均使用了描述逻辑中现有的大部分优化技术，诸如概念标准化和简化，吸收与阻塞技术，语义分支搜索，依赖制导回溯等. 两者的差异如下：

(1) 在预处理过程中，执行完标准化和吸收之后，FaCT++继续执行循环消除和同义词替换步骤以此进一步简化知识库.

(2) 在可满足性检测过程中，Pellet 使用的是通常的 top-down 方法进行 tableaux 扩展，而 FaCT++则使用 ToDo 列表来控制扩展规则的应用，根据排序启发式策略选取优先级最高的节点首先扩展. 此外，Pellet 采用了局部回溯方法，该方法维持一个有效信息的依赖集合，防止回溯过程中的信息丢失，并有效避免重复使用 tableaux 规则. 而 FaCT++则着眼于降低 tableaux 扩展的复杂性，使用了布尔常量传播优化技术来减少非确定性规则的使用.

(3) 在分类过程中，Pellet 通过采用基于名词的模型融合方法去发现明显的非包含关系. 而 FaCT++注重减少需要执行的包含关系检测数量，使用 TBox 公理的语法结构来优化所计算的分类结构顺序，例如，给定公理 $C \sqsubseteq D$，FaCT++会延迟把 C 添加到分类结构中直到所有出现在 D 中的概念都完成分类为止，从而避免对新添加进去的概念马上进行包含检测.

JFact 是 FaCT++的 Java 实现并具有扩展的数据类型支持[25]，两者具有相同的推理机制，但 JFact 的开发工作还正在进行中，其性能的巨大差异取决于具体的推理任务，另外，JFact 仍然存在一些尚未发现的错误和漏洞[26]，导致其在求解 Mups 问题上，特别是涉及到较大规模的测试数据时，性能下降很快. 除此之外，Pellet 还采用名词吸收技术来处理枚举类型，并对 ABox 的合取查询进行了一系列优化. 由于 MupsBM-$\mathcal{ALC}$数据中不涉及枚举类型和 ABox 的合取查询，Pellet 无法在评测实验中发挥这些优势.

与 Pellet、FaCT++以及 JFact 不同，Hermit 是基于 hypertableau 算法开发的. 对于基于标准 tableau 算法的推理机来说，概念$\neg A \sqcup B$ 的可满足性检测过程中，使用$\sqcup$-规则会导致非确定性情况的出现. Hermit 通过重写技术减少该情况出现，它将这种概念转换为 $A \sqsubseteq B$ 的形式，因此能够使用吸收类型的优化，这是标准 tableau 算法推理机所不具备的. 此外，Hermit 还利用了个体重用(individual reuse)技术，当扩展存在限定概念 $\exists R.C$ 时，Hermit 首先考虑的是能否重用已经存在于 C 标签中的个体来构建模型，如果不能，才考虑引进一个新的个体. 该方法能够减少模型生成的规模，特别是针对大型的复杂本体效果明显，由于 MupsBM 未涉及 ABox 中的个体断言，也导致 Hermit 此项优势未能发挥.

与上述推理算法均不同，TrOWL 基于近似方法和遗忘策略设计其推理机制，它使用语法近似技术将 OWL2 转化为 OWL2-EL 实现 TBox 推理. 在转化过程中会付出一些时间代价，此外，TrOWL 采用了遗忘策略，该策略基于原本体构建一个仅限于某一特定概念的小型本体来简化原本体，从而排除那些不必要的概念. 然而，遗忘策略的适用与否受限于具体的本体内部结构，因此存在某些现实本体或是 MupsBM 测试数据，不一定适用于遗忘策略的情形.

6　结束语

使用推理机对某个不一致术语集进行调试时，为了评测不同推理机求解 Mups 的效率高低，本文在构建术语集依赖关系图模型基础之上，从概念之间的依赖关系角度出发，定义语义依赖度、语义簇与聚合度、依赖度分布 3 个指标反映术语集的复杂程度；又从术语集的不一致性出发，探讨影响求解 Mups 难易程度的那些因素，将其作为不可满足概念复杂程度的评价指标，基于这些复杂性评价指标，设计出评测推理机求解 Mups 性能高低的标准检查程序 MupsBen. MupsBen 在可满足概念数目、不可满足概念数目、语义簇数目、最大语义依赖度、冲突公理集最大基数和冲突公理最大基数 6 个复杂性指标的指导下，生成 Benchmark 测试数据，评测实验调用 Pellet、Hermit、FaCT++、JFact 和 TrOWL 这 5 种推理机求解 Mups. 结果显示，从整体上看，对于特定推理机，其性能具有随测试数据的结构复杂程度的增大而降低的趋势；对于不同推理机，由于

其内在推理机制与优化策略的差别,在不同复杂度指标下表现出不同的性能差异,特别指出的是,在大规模本体数据的背景下,本体复杂度较高时,某些推理机无法求出问题的解.因此,MupsBen 能够提供一个针对 Mups 推理问题如何评测推理机性能优劣问题的 Benchmark,根据评测结果,一方面有助于推理机用户为不同结构特点的本体选择合适的推理机为本体调试提供依据;另一方面有助于推理机开发者了解影响推理问题难易程度的具体因素,在此基础之上改进推理机的设计.

特别指出的是,由于本文的测试数据是$\mathcal{ALC}$表达能力级别的 TBox 数据,所以各个推理机性能的评测结果只针对求解$\mathcal{ALC}$ TBox 数据的 Mups 具有参考价值,随着 TBox 测试数据表达能力的增加,各个推理机在求解 Mups 问题时的性能对比会随之发生变化.例如,Pellet 采用名词吸收技术来处理枚举类型,其优势在$\mathcal{ALC}$数据上则无法体现出来.另外对于那些在 ABox 推理上占优势的推理机也无法体现出其性能.

本文构造出的针对某一特定推理问题的 Benchmark 只是提供了解决此类问题的一种思路,其他研究者可以借鉴该思路开发自己的 Benchmark,下一步的工作将着眼于提高测试数据的表达能力,设计出语义更丰富的并针对更多推理问题的 Benchmark.

参 考 文 献

[1] Ouyang Dantong, Cui Xianji, Ye Yuxin. Mapping integrity constraint ontology to relational database. Journal of China Universities of Posts and Telecommunications, 2010, 17(6): 113-121

[2] Ye Yu-Xin, Ouyang Dan-Tong. Semantic-based focused crawling approach. Journal of Software, 2011, 22(9): 2075-2088(in Chinese)
(叶育鑫,欧阳丹彤. 基于语义的主题爬行策略. 软件学报, 2011, 22(9): 2075-2088)

[3] Guo Yuanbo, Pan Zhengxiang, Heflin J. LUBM: A benchmark for OWL knowledge base systems. Web Semantics: Science, Services and Agents on the World Wide Web, 2005, 3(2): 158-182

[4] Ma Li, Yang Yang. Towards a complete OWL ontology Benchmark//Proceedings of the 3rd European Semantic Web Conference. Budva, Montenegro, 2006: 125-139

[5] Li Yingjie, Yu Yang, Heflin J. A multi-ontology synthetic benchmark for the semantic web//Proceedings of the International Workshop on Evaluation of Semantic Technologies. Shanghai, China, 2010: 40-51

[6] Imprialou M, Stoilos G. Benchmarking ontology-based query rewriting systems//Proceedings of the 26th Conference on Artificial Intelligence. Toronto, Canada, 2012: 779-785

[7] Raunich S, Rahm E. Towards a benchmark for ontology merging//Proceedings of the Workshops on the Move to Meaningful Internet Systems. Rome, Italy, 2012: 124-133

[8] Ferrara A, Montanelli S. Benchmarking matching applications on the semantic web//Proceedings of the 8th Extended Semantic Web Conference. Crete, Greece, 2011: 108-122

[9] Meilicke C. MultiFarm: A benchmark for multilingual ontology matching. Web Semantics: Science, Services and Agents on the World Wide Web, 2012, 15(1): 62-68

[10] Rosoiu M. Ontology matching benchmarks: Generation and evaluation//Proceedings of the 6th ISWC Workshop on Ontology Matching. Bonn, Germany, 2011: 73-84

[11] Meyer T, Lee K, Booth R. Finding maximally satisfiable terminologies for the description logic $\mathcal{ALC}$//Proceedings of the 21st National Conference on Artificial Intelligence. Boston, USA, 2006: 269-274

[12] Schlobach S, Huang Zhisheng. Debugging incoherent terminologies. Journal of Automated Reasoning, 2007, 39(3): 317-349

[13] Kalyanpur A, Parsia B. Debugging unsatisfiable classes in OWL ontologies. Journal of Web Semantics, 2005, 3(4): 268-293

[14] Kalyanpur A, Parsia B. Repairing unsatisfiable concepts in OWL ontologies//Proceedings of the 3rd Europen Semantic Web Conference. Budua, Yogosialia, 2016: 170-184

[15] Kalyanpur A, Parsia B, Horridge M. Finding all justifications of OWL DL entailments//Proceedings of the 6th International Semantic Web Conference. Busan, Korea, 2007: 267-280

[16] Bail S, Parsia B, Sattler U. Diversity of reason: Equivalence relations over description logic explanations//Proceedings of the 25th International Workshop on Description Logics. Rome, Italy, 2012: 48-58

[17] Aslani M, Haarslev V. TBox classification in parallel: Design and first evaluation//Proceedings of the 23rd International Workshop on Description Logics. Waterloo, Canada, 2010: 336-347

[18] Stoilos G. Repairing incomplete reasoners//Proceedings of the 24th International Workshop on Description Logics. Barcelona, Spain, 2011: 411-421

[19] Sirin E, Parsia B. Pellet: A practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web, 2007, 5(2): 51-53

[20] Shearer R, Motik B. HermiT: A highly-efficient OWL reasoner//Proceedings of the 5th International Workshop on OWL: Experiences and Directions. Karlsruhe, Germany, 2008: 1-10

[21] Tsarkov D, Horrocks I. FaCT++ description logic reasoner: System description//Proceddings of the 3rd International Joint Conference on Automated Reasoning. Seattle, USA, 2006: 292-297

[22] Thomas E. TrOWL: Tractable OWL 2 reasoning infrastructure //Proceedings of the 7th Extended Semantic Web Conference. Heraklion, Greece, 2010: 431-435

[23] Schlobach S, Cornet R. Non-standard reasoning services for the debugging of description logic terminologies//Proceedings of the 18th International Joint Conference on Artificial Intelligence. Acapulco, Mexico, 2003: 355-360

[24] Kalyanpur A. Debugging and Repair of OWL Ontologies [Ph. D. dissertation]. Department of Computer Science, University of Maryland, Maryland, USA, 2006

[25] Gonçalves R, Bail S. OWL reasoner evaluation (ORE) workshop 2013 results: Short report//Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE-2013). Ulm, Germany, 2013: 1-18

[26] Tsarkov D, Palmisano I. Chainsaw: A metareasoner for large ontologies//Proceedings of the OWL Reasoner Evaluation Workshop(ORE 2012). Manchester, UK, 2012: 19-27

OUYANG Dan-Tong, born in 1968, Ph. D., professor, Ph. D. supervisor. Her current research interests include model-based diagnosis and automated reasoning.

ZHANG Yu, born in 1982, Ph. D. candidate. His current research interests include description logic and ontology debugging.

YE Yu-Xin, born in 1981, Ph. D., associate professor. His current research interests include semantic web and ontology engineering.

Background

Ontology building is an important aspect of the study in semantic web. Due to a lack of knowledge and experience, a new ontology modeler is easy to make errors. Unsatisfiable concepts are a fundamental modeling errors, the existence of unsatisfiable concepts indicates that the formal definition of ontology is incorrect. Therefore, how to find minimal unsatisfiability preserving sub-terminologies (Mups) is the mainly means of ontology debugging. Currently, various reasoners have been developed for several standard reasoning tasks, namely classification, concept satisfiability checking and consistency checking.

Finding all the Mups is an inference task by using reasoners. Most of reasoners, such as Pellet, Hermit, TrOWL, FaCT++ and JFact, can support the task of finding Mups in an incoherent ontology. One of the primary goals of the study is how to choose an appropriate reasoner for a reasoning task. Most existing approaches are in the field of knowledge base inference, ontology query, ontology merging, ontology matching. But so far, there is a lack of available benchmarks for Mups reasoning.

Therefore, Our goal is to provide a benchmark for evaluating reasoner for calculating Mups. A key benefit of our benchmarking approach is that we define three metrics of semantic dependence, semantic cluster and dependence distribution to reflect the complexity of terminology on the basis of terminology dependency graph model, and we define two metrics of max cardinal of conflict axiom set and max cardinal of conflict axiom to reflect the complexity of incoherent terminology. Then we develop a Mups Benchmark (MupsBen) to evaluate the performances of reasoners for calculating Mups based on those metrics.

This work was supported by the National Natural Science Foundation of China (Nos. 61133011, 41172294 and 61170092, 61272208), the Science and Technology Development Plan of Jilin Province (No. 201201011). The focus is on identifying features of incoherent ontology that make finding Mups hard or easy for different reasoners, which can help the ontology modelers to get further insights into the features of ontology for debugging and contribute to the improvement of reasoners according to the evaluation results.