4. 从 $R(x)$中获得α_k的不可满足依赖标签 $L(\alpha_k)=(\varepsilon_k,\alpha_k,\varepsilon_{k+1})$
5. **if** Y 形如 $Y_1\cup\ldots\cup Y_m$ **then**
6. **for each** Y_j **in** Y
7. P_Y=***Partition*1**(Y_j,$L(\alpha_k)$)
8. **if** X 形如 $X_1\cap\ldots\cap X_n$ **then**
9. **for each** X_i **in** X
10. P_X =***Partition*1**(X_i,$L(\alpha_k)$)
11. **else** $P_Y=\{Y\}$, $P_X=\{X\}$
12. **for each** y **in** P_Y and **each** x **in** P_X
13. *S.add*($y\subseteq x$)
14. **end for**
15. **end for**
16. $S_{\min}$=**Minimality**(S)
17. **return** $S_{\min}$ //输出修复集

在算法 6 中,当待分割公理左侧是析取式时,则对该析取式的每一部分,调用函数 *Partition*1(Y_j,$L(\alpha_k)$)进行分割(第 5 行~第 7 行).当待分割公理右侧是合取式时,则对该合取式的每一部分,调用函数 *Partition*1(X_i,$L(\alpha_k)$)进行分割(第 8 行~第 10 行).其他情况则保留原表达式不变(第 11 行).分割完成后对于获取的分割集 P_Y 与 P_X 中的每个元素 y 与 x,重新组合成新的公理(第 12 行~第 13 行).最后,再对获取的分割集进行一次极小化操作(第 16 行).

需特别说明的是,对于公理右边的析取形式与公理左边的合取形式没有进行分割操作.这是因为分割的目的是排除与不可满足性无关的那些部分.从不可满足性的根源考虑,公理右边的析取式的各部分必须同时与不可满足性有关,以及公理左边的合取式的各部分必须同时与不可满足性有关,这样的公理才能出现于 MUPS 中.

函数.*Partition*1(S).

输入:S;

输出:$P_{X/Y}$.

1. **if** $\varepsilon_k\subseteq sig(Y_j)$且$\varepsilon_{k+1}\subseteq sig(X_i)$,或$\varepsilon_{k+1}\subseteq sig(Y_j)$且$\varepsilon_k\subseteq sig(X_i)$ **then**
2. **if** X_i 或 Y_j 形如 C,$\neg$或$\exists R.C$ **then**
3. $P_X=\{X_i\}$或 $P_Y=\{Y_j\}$
4. **if** X_i 或 Y_j 形如$\forall R.C$ **then**
5. P_X 或 P_Y=***Partition*2**($\forall R.C$,$L(\alpha_i)$) //求出$\forall R.C$ 的分割集
6. **return** P_X 或 P_Y

在分割函数 *Partition*1 中,首先判断待分割的 Y_j 或 X_i 是否存在 $L(\alpha_k)$中的依赖概念,若存在,则进行分割,在分割过程中,保留形如 C,$\neg C$ 或$\exists R.C$ 的 X_i 或 Y_j,而对形如$\forall R.C$ 的 X_i 与 Y_j,则进一步调用函数 *Partition*2 进行分割.在分割函数 *Partition*2 中,首先存储角色 R 于 *prop* 中,然后针对合取式 C 递归调用 *Partition*2,继续进行分割.

函数.*Partition*2($\forall R.C$,$L(\alpha_i)$).

输入:$\forall R.C$,$L(\alpha_i)$;

输出:P.

1. *prop*=$\{R\}$
2. **if** C 形如 C_k,$\neg C_k$ 或$\exists R.C_k$ 且$\varepsilon_k\subseteq sig(C_k)$或$\varepsilon_{k+1}\subseteq sig(C_k)$, **then**
3. $P=\{\forall R.C\}$
4. **if** C 形如 $C_1\cap\ldots\cap C_n$, **then**
5. *P=Partition*2($\forall R.C_k$,$L(\alpha_i)$)
6. **return** P

极小化函数 Minimality(S)是为了排除那些存在多个相同否定概念的公理.例如公理 $B\subseteq A\cap\neg A\cap\exists R.\neg A$,对其分割后得到 $S=\{B\subseteq A,B\subseteq\neg A,B\subseteq\exists R.\neg A\}$,$B\subseteq\exists R.\neg A$ 之所以能存在于 S 中,是因为只有当发现某个公理存在否定概念时,才开始构造与该否定概念有关的冲突路径,但是无法区分该否定概念究竟是 $B\subseteq\neg A$ 还是 $B\subseteq\exists R.\neg A$ 中的否定概念.这一现象不会影响本体调试结果,但会造成修复集存在冗余的公理.因此需要进行一次极小化操作.具体方法是对每一个存在否定概念的公理(第 1 行~第 3 行),将它从修复集中删除(第 4 行),然后检测该修复集中的不可满足概念集是否有变化(第 5 行),若无变化,则该公理是冗余公理,删除即可(第 6 行).

函数.Minimality(S).

输入:S;

输出:$S_{\min}$.

1. **for each** $\neg\theta$ **in** $sig(S)$
2. **for each** α=$y\subseteq x$ **in** S
3. **if** $\neg\theta\in sig(y)$或 $sig(x)$, **then**
4. $S'=S.remove(\alpha)$
5. **if** $U_S=U_{S'}$ **then** //若 S 与 S'中的不可概念集 U_S 与 $U_{S'}$相同
6. $S_{\min}=S.remove(\alpha)$ //删除冗余公理α
7. **return** $S_{\min}$

考虑本节开始的引例 $T=\{\alpha_1:C\subseteq A,\alpha_2:A\subseteq E_1\cap D_1,\alpha_3:D_1\cup E_2\subseteq D_2,\alpha_4:D_2\subseteq E_3\cap\neg A,\alpha_5:E_1\subseteq E_3.\}$通过上述方法得到的修复集 $S=\{A\subseteq D_1,D_1\subseteq D_2,D_2\subseteq\neg A\}$,极小化 S 后仍是$\{A\subseteq D_1,D_1\subseteq D_2,D_2\subseteq\neg A\}$.

设 $M_T(C)$是一个有 m 个公理的 MUPS,求 $M_T(C)$的冲突路径的复杂度是 km,其中,k 是冲突路径的条数.这是因为在使用本文的方法构造 $M_T(C)$的冲突路径时,有效地避免了循环操作,从而大大降低了求解冲突路径的复杂度.接下来,从冲突路径上出现的所有概念中筛选出不可满足概念,从而将冲突路径变为不可满足概念依赖路径,对于出现 n 个概念的冲突路径,判断每个概念是否属于不可满足概念集合(不可满足概念集合在调试阶段已经得到),一共需要 n 次判断.所以,算法 5 的时间复杂度为 $k(m+n)$.在获取修复集的过程中,对于有 m 个公理的 $M_T(C)$,需要从不可满足概念依赖路径中获取每个公理的不可满足概念的依赖标签,该过程的时间复杂度为 m.这是因为,在构造依赖路径时,路径上的每一个节点就是一个依赖标签,一旦匹配成功,就直接可以读取该标签.然后根据依赖标签对公理 $Y\subseteq X$ 进行分割,在此过程中,需要进行公理类型的判断:如果 Y 形如 $Y_1\cup\ldots\cup Y_p$,则需要 p 次分割操作,如果 X 形如 $X_1\cap\ldots\cap X_q$,则需要 q 次分割操作,而每次操作都需要调用分割函数 1 和分割函数 2,每个分割函数都需要进行 2 次判断.最后是极小化过程,设修复集的公理个数为 s,修复集中否定概念的个数为 t,此时需要进行 st 次可满足性检测.所以,算法 6 的时间复杂度为 $2mpq+st$.

4 实验与分析

本文实验是在 PC 机 Windows 10 操作系统(Intel(R) Core(TM) i7-6700 CPU @ 3.40GHZ,16.0G 内存)下运行的.对本体的解析和操作采用了 OWLAPI(http://owlapi.sourceforge.net/),使用推理机 Pellet(https://github.com/stardog-union/pellet)对概念进行可满足性检测,最大 JVM 堆设置为 3.0G.超时界限设定为 60 分钟.实验数据、CP 算法源码与结果由 http://www.zhyweb.cn/cpdebrep/index.php 下载.

本文实验的数据来源自本体库(http://www.cs.ox.ac.uk/isg/ontologies/lib/),鉴于所提出的调试与修复策略是基于不协调的 ALCOI-TBox 本体的,本文使用文献[26]所采取的处理方式,将不符合 ALCOI-TBox 的公理删去,并以随机的方式从本体中选择两个概念构成不相交公理添加进原本体中,从而生成不协调本体.例如,对于协调的本体 $T=\{A\subseteq B,B\subseteq C,B\subseteq D\}$,通过添加公理 $C\subseteq\neg D$ 从而导致 A 和 B 不可满足.表 1 列出了实验数据的主要属性.$|T|$是本体规模,$|U|$是不可满足概念个数,$MN_{\max}$ 与 ML_{avg} 分别是 MUPS 最大基数与最大长度,MN_{avg} 与 ML_{avg} 分别是 MUPS 的平均基数与平均长度.

Table 1 Properties of incoherent TBoxes used in the experiments

表 1 实验中不协调 TBox 的主要属性

ID	TBox	$\|T\|$	$\|U\|$	MN_{max}	ML_{max}	MN_{avg}	ML_{avg}
T_1	OBI10	3 968	103	8	19	2.330	5.167
T_2	PATO	3 489	158	4	8	1.177	3.409
T_3	NIF	3 460	133	10	12	3.519	5.447
T_4	OBO09	34 367	59	4	10	1.085	2.953
T_5	OBO12	51 782	24	7	8	2.375	3.018
T_6	Cellular	6 190	64	10	7	2.766	4.305
T_7	OBI09	2 017	22	3	6	1.182	3.423
T_8	SAO	1 112	241	3	9	1.075	6.139
T_9	SWO	2 624	166	4	4	1.560	2.807
T_{10}	OBOGO	37 615	40	3	7	1.750	3.700
T_{11}	Anatomy	36 157	35	4	15	1.257	5.045
T_{12}	MAO	1 619	20	1	5	1.000	2.200
T_{13}	FLU	1 326	65	4	33	1.600	11.692
T_{14}	Bockman	3 043	25	2	4	1.160	2.138
T_{15}	Cavender	1 623	23	2	7	1.130	3.731

4.1 基于冲突路径的本体调试实验

实验针对黑盒法的 4 种优化算法和 3 个调试工具展开.黑盒优化算法包括扩张阶段的选择函数方法(记作 *S*)与模块化方法(记作 *M*)以及收缩阶段的滑动窗口方法(记作 *W*)与分治方法(记作 *D*).对比实验中将两个阶段的优化方法组合起来构成 4 种算法(例如选择函数与滑动窗口组合算法记作 *SW*).针对每一种组合算法,采用本文的基于冲突路径(记作 *CP*)的策略进行优化,优化后的算法记作 CPSW.3 个调试工具包括 Radon[19]、Swoop[24]和 Protégé(http://www.cs.ox.ac.uk/isg/ontologies/lib/).针对每一个调试工具,采用 CP 策略获取冲突集,将冲突集作为调试工具的输入.图 1 显示的是 14 种算法求解所有不可满足概念的所有 MUPS 的时间的对比情况.

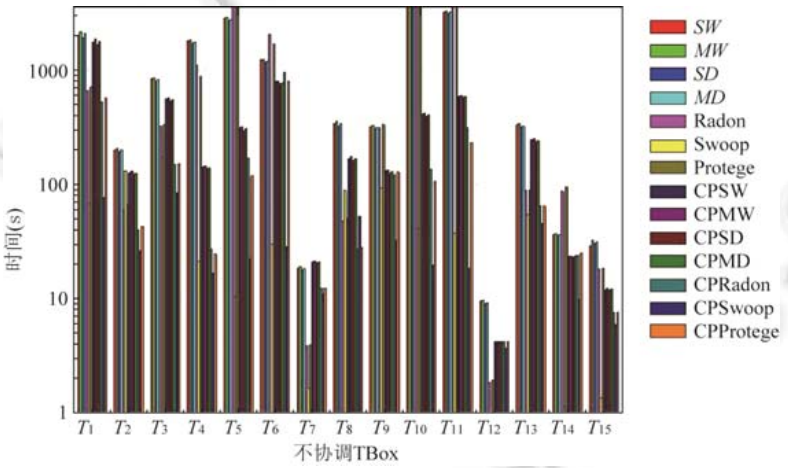

Fig.1 Compared run time of fourteen algorithms for computing MUPS

图 1 14 种算法求解 MUPS 的时间对比

图 1 中,除了 T_7 之外,基于 CP 优化策略的组合算法均优于未使用 CP 优化策略的组合算法.除了 T_7 和 T_{12} 之外,基于 CP 优化策略的调试工具均优于未使用 CP 优化策略的调试工具.这是因为,相对于黑盒法求解 MUPS 频繁地调用推理机进行可满足性检测,基于冲突路径获取冲突集的方法仅仅在判定获取的否定路径和互补路径的公理集是否协调时才调用一次推理机,因而所需时间较少,这是冲突路径优化策略性能较好的主要原因.表 2 对 CP 优化策略获取冲突集的属性进行了统计,$|CP|$是互补概念对的个数,$|Pn|$与$|Pp|$分别是否定路径与肯定路径中公理的数目,$|Q|$是冲突集里公理的数目,$|Q/T|$是冲突集占原本体的百分比,*Time* 是获取冲突集的时间(单位:s).

Table 2 Properties of obtaining clash set based on CP approach

表 2 CP 优化策略获取冲突集的属性统计

TBox	$\|T\|$	$\|CP\|$	$\|U\|$	$\|Pn\|$	$\|Pp\|$	$\|Q\|$	$\|Q/T\|$ (%)	*Time*(s)
T_1	3 968	274	103	3 941	3 666	3 670	92.49	22.001
T_2	3 489	269	158	3 465	3 196	3 176	91.03	6.937
T_3	3 460	261	133	3 413	3 152	2 211	63.90	5.311
T_4	34 367	78	59	10 436	6 158	5 290	15.39	4.520
T_5	51 782	99	24	19 003	25 387	12 466	24.07	18.674
T_6	6 190	120	64	5 726	4 658	4 439	71.71	10.917
T_7	2 017	160	22	1 816	1 656	1 658	82.20	9.490
T_8	1 112	206	241	1 024	832	816	73.38	3.338
T_9	2 624	159	166	1 508	1 224	1 555	59.26	1.420
T_{10}	37 615	210	40	22 902	15 824	6 418	17.06	13.63
T_{11}	36 157	272	35	19 775	15 426	13 030	36.04	12.626
T_{12}	1 619	247	20	1 567	1 317	758	46.82	3.267
T_{13}	1 326	275	65	1 301	1 026	1 047	78.96	5.717
T_{14}	3 043	94	25	3 014	2 911	1 102	36.21	8.804
T_{15}	1 623	264	23	1 595	1 332	509	31.37	4.89

综合分析图 1 与表 2 的实验结果,可以得出如下结论.

(1) 在本体规模较大、不可满足概念较多以及 MUPS 结构较复杂的情况下(体现为表 1 的后 4 个属性),黑盒法求解 MUPS 需要耗费大量的时间.例如使用 *SW* 方法调试 T_{12} 本体的时间为 9.531s,而调试 T_5 本体的时间则高达 2 862.556s,而 T_{10} 本体的调试时间则超过 60 分钟.

(2) 当使用 CP 优化策略时,如果 MUPS 求解过程是基于一个较小规模的冲突集之上,则效率较高.例如,对于规模较大的本体(例如 T_4,T_5 与 T_{10}),所得到的冲突集相对于原本体较小(分别是 15.39%、24.07%与 17.06%),则 CP 效率较高,例如 T_4 本体,*SW*=1818.296s,而 *CPSW*=142.778s.如果冲突集接近于原本体,则 CP 方法效率就会受到一定的影响,例如,T_1 本体的求解时间分别为 *SW*=2034.249s 和 *CPSW*=1755.991s,这是因为冲突集占原本体的比例高达 92.49%.

(3) 当获取冲突集所耗费的时间较多时,则 CP 优化策略的性能就无法有效发挥.例如,对于 T_{77},*SW* 的求解时间是 18.407s,而 *CPSW* 的求解时间是 20.961s,这是由于 CP 方法获取冲突集消耗了 9.490s 的时间,而基于冲突集求解 MUPS 的时间是 11.471s.相对于求解 MUPS 所需要的时间,获取冲突集的时间较多,因而影响了 CP 优化算法的性能.

(4) 在影响黑盒法求解效率的众多因素中,本体规模的影响相对较大,而 CP 优化策略所获取的冲突集仅由冲突路径所决定,因而基本上不受本体规模的影响.例如 T_4,本体规模为 34 367,而冲突集规模仅为 5 290,而获取冲突集的时间仅为 4.520s,因而基于 CP 优化的黑盒法在 5 290 规模量级的冲突集上求解 MUPS 的效率较高.

本文提出的基于冲突路径的本体调试方法可以说是一个基于模块的本体调试方法,如下实验比较了基于语法局部(syntactic locality module)的模块化方法[14]和本文方法抽取的模块中公理的数量.

设 T 有 n 个不可满足概念,对于不可满足概念 u_i,在采用基于模块的黑盒调试方法求解 MUPS(T,u_i)时,在扩张阶段,抽取与 u_i 有关的基于语法局部的模块,记为 $M(u_i)$,再在收缩阶段将 $M(u_i)$削减成极小集合.

设 $M(u_i)$的公理数量为 $S_{M(u_i)}$, 那么 n 个不可满足概念需要抽取的模块大小则为 $M=\sum_1^n S_{M(u_i)}$.

表 3 中第 2 行统计的是基于原本体所抽取的所有不可满足概念的模块中公理的总数量,第 3 行统计的是:基于各个本体的冲突集所抽取的所有不可满足概念的模块中公理的总数量.

Table 3 Modules extracted by Modularity and CPModularity methods for all unsatisfiable concepts

表 3 Modularity 与 CPModularity 方法抽取所有不可满足概念的模块统计结果

方法	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
Modularity	13 625	11 878	11 074	2 298	1 198	4 128	1 857	16 798	3 623	7 038	5 896	116	4 604	230	194
CPModularity	11 992	9 035	8 557	518	146	2 567	1 744	16 432	1 194	305	341	45	4 390	113	182

表 3 显示,基于冲突集所抽取的模块中公理的数量确实少于基于原本体所抽取的模块中公理的数量.

需要说明的是:本文获取冲突集这一过程不是取代基于“扩-缩”策略的黑盒法的扩张阶段,而是尽可能地为黑盒调试算法和调试工具提供一个较小规模的输入.这是因为:黑盒法对本体规模非常敏感,一个较小规模的输入本体对黑盒法的效率提升是有很大帮助的.

4.2 基于冲突路径的本体修复实验

虽然基于DL-Lite描述逻辑的本体修复方法较多,但这些方法均利用DL-Lite特殊的语法特征构造DL-Lite逻辑闭包图完成本体修复,这类方法无法处理任何非 DL-Lite 的本体.因而本文基于 ALCOI 描述逻辑的本体修复实验无法与 DL-Lite 本体的修复方法进行对比.而对于非 DL-Lite 本体的修复方法,最具代表性的是基于根不可满足概念实现细粒度本体修复的方法(记作 RSRepair[12]),与此对应的一套算法包括:基于依赖跟踪的方法获取根不可满足概念,基于公理分割的方法对 MUPS 中的所有公理进行分割获取 MUPS 的分割集,以及基于分割集再次求解MUPS的细粒度修复.这一套算法与本文提出的基于冲突路径的修复方法(记作CPRepair)的出发点和目标都是相同的,差别在于如下 3 点:一是获取根不可满足概念的方式,二是公理分割的对象,三是公理分割的方式,因而具有较强的可比性.表 4 列出了 RSRepair 与 CPRepair 实验对比结果.

Table 4 Compared results of RSRepair and CSRepair for ontology repairing (ms)

表 4 RSRepair 与 CPRepair 本体修复对比实验结果 (毫秒)

方法	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
RSRepair	1 844	91	129	177	51	27	53	55	32	68	722	175	3579	122	870
CPRepair	174	83	**693**	76	45	**105**	47	**135**	**151**	53	71	68	263	57	56

表 4 显示,除了 T_3,T_6,T_8,T_9 这 4 个本体外,CPRepair 均优于 RSRepair.为了深入 RSRepair 方法内部探查其各个环节对修复效率的影响,表 5 统计了基于依赖跟踪方法获取根不可满足概念的时间($Time_{root}$),基于公理分割方法获取 MUPS 分割集的时间($Time_{splitting}$)以及基于分割集再次求解 MUPS 的时间($Time_{MUPS}$).由表 5 可知,前两个环节所花费的时间较少,时间基本上都耗费在第 3 个环节,这是由于,基于分割集再次求解 MUPS 的过程需要多次调用推理机进行可满足性检测.本文CPRepair方法的不同之处在于:一是从MUPS中获取冲突路径,再从冲突路径中筛选出根依赖路径从而确定根不可满足概念;二是公理分割仅仅针对具有依赖概念的那部分公理.整个过程只有在筛选根依赖路径以及对修复集进行极小化操作时才调用推理机,因而效率较高.特别是 T_1,T_{11},T_{13},T_{15} 这 4 个本体,修复效率比 RSRepair 高达 10 多倍.对于 T_6 与 T_9,一方面是分割集较小,另一方面是分割集基本接近最终的MUPS求解结果,因而RSRepair所耗费的时间很少.需要补充说明的是,虽然某些不可满足概念存在着根与派生两种属性(见第 3.1 节),但是 RSRepair 在对 MUPS 中的所有公理进行分割之后,就可以将这两种属性分割开来,从而可以将该不可满足概念的两种属性同时显示出来.

Table 5 Time of three algorithms for RSRepair (ms)

表 5 RSRepair 方法的 3 种算法运行时间 (毫秒)

时间	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8	T_9	T_{10}	T_{11}	T_{12}	T_{13}	T_{14}	T_{15}
$Time_{root}$	6	5	7	5	3	3	9	5	8	3	3	6	5	13	6
$Time_{splitting}$	9	5	4	8	5	4	5	4	4	5	7	6	6	6	5
$Time_{MUPS}$	1 829	81	118	164	43	20	39	46	20	60	712	163	3 568	103	859
All time	1 844	91	129	177	51	27	53	55	32	68	722	175	3 579	122	870

获取了根不可满足概念的修复集之后,就可以提交给领域专家进行修复,只要完成了根不可满足概念的修复,就完成了整个不协调本体的修复.

5 结束语

本文对不协调本体调试与修复的黑盒法进行了研究,基于“扩张-收缩”策略的黑盒法及其优化方法独立于特定的推理机而应用十分广泛.然而,它们在求解 MUPS 过程中很容易选出与不协调无关的冗余公理而影响本体调试的效率.针对这一问题,提出了一种基于冲突路径的调试与修复策略,该策略通过构造与 4 种基本冲突模

式所对应的冲突路径,能够获取与不协调密切相关的冲突集.在此过程中,由于推理机的调用次数很少,所以获取冲突集的时间耗费较少.接下来,将 MUPS 的求解过程限定在该冲突集上能够较大幅度地减少参与黑盒法求解的术语集规模,这是效率得以提高的主要原因.进一步地,借助于冲突路径的良好结构,获得不可满足概念依赖路径并基于该路径制定出不可满足概念的修复策略,该策略能够获取精确的修复集,并且能够避免本体信息内容的损失.需要特别说明的是,本体的描述逻辑语言不同,冲突路径的构造方式也随之会变化,下一步的工作将针对更高表达能力的描述逻辑本体构造其对应的冲突路径进行本体调试与修复.

References:

[1] Baader F, Calvanese D, McGuinness D, Nardiand D, Patel-Schncider PF. Basic Description Logics. 2nd ed., Cambridge: Cambridge University Press, 2007. 47–100.

[2] Li P, Jiang YC, Wang J. Modular ontology reuse based on conservative extension theory. Ruan Jian Xue Bao/ Journal of Software, 2016,27(11):2777–2795 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4920.htm [doi: 10.13328/j.cnki.jos.004920]

[3] Cui XJ, Ouyang DT, Ye YX. Integrity constraint validation for ontology using modularization. Journal of Information & Computational Science, 2014,11(9):2705–2713. [doi: 10.12733/jics20103476]

[4] Ouyang DT, Qu JF, Ye YX. Extending training set in distant supervision by ontology for relation extraction. Ruan Jian Xue Bao/Journal of Software, 2014,25(9):2088–2101 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4638.htm [doi: 10.13328/j.cnki.jos.004638]

[5] Schlobach S, Huang Z, Cornet R, Van Harmelen F. Debugging incoherent terminologies. Journal of Automated Reasoning, 2007,39(3):317–349. [doi: 10.1007/s10817-007-9076-z]

[6] Meyer T, Lee K, Booth R, Pan J.Z. Finding maximally satisfiable terminologies for the description logic ALC. In: Proc. of the 21st AAAI. American Association for Artificial Intelligence, 2006. 269–274.

[7] Ouyang DT, Su J, Ye YX, Cui XJ. The ontology debugging method based on concept R-MUPS. Ruan Jian Xue Bao/Journal of Software, 2015,26(9):2231–2249 (in Chinese with English abstract). http://www.jos.org.cn/1000-9825/4735.htm [doi: 10.13328/j.cnki.jos.004735]

[8] Schlobach S, Cornet R. Non-Standard reasoning services for the debugging of description logic terminologies. In: Proc. of the 18th Int'l Joint Conf. on Artificial Intelligence. 2003. 355–360.

[9] Baader F, Penaloza R. Automata-Based axiom pinpointing. In: Proc. of the 4th IJCAR. Heidelberg: Springer-Verlag, 2008. 226–241.

[10] Baader F, Penaloza R, Suntisrivaraporn B. Pinpointing in the description logic EL+. In: Proc. of the 20th DL. Tilburg: Sun SITE Central Europe CEUR-WS, 2007. 171–178.

[11] Baader F, Suntisrivaraporn B. Debugging snomed CT using axiom pinpointing in the description logic EL+. In: Proc. of the 3rd KR-MED. Tilburg:Sun SITE Central Europe CEUR-WS, 2008. 1–7.

[12] Kalyanpur A. Debugging and repair of OWL ontologies [Ph.D. Thesis]. Maryland: University of Maryland, 2006.

[13] Kalyanpur A, Parsia B, Horridge M, Sirin E. Finding all justifications of OWL DL entailments. In: Proc. of the 6th ISWC. Heidelberg: Springer-Verlag, 2007. 267–280. [doi: 10.1007/978-3-540-76298-0_20]

[14] Cuencagrau B, Ian H, Yevgeny K, Ulrike S. Just the right amount: Extracting modules from ontologies. In: Proc. of the 16th Int'l World Wide Web Conf. New York: Association for Computing Machinery, 2007. 717–726. [doi: 10.1145/1242572.1242669]

[15] Ji Q, Qi GL, Haase P. A relevance-directed algorithm for finding justifications of DL entailments. In: Proc. of the 4th ASWC. Heidelberg: Springer-Verlag, 2009. 306–320. [doi: 10.1007/978-3-642-10871-6_21]

[16] Suntisrivaraporn B, Ji Q, Haase P. A modularization-based approach to finding all justifications for OWL DL entailments. In: Proc. of the 3rd ASWC. Heidelberg: Springer-Verlag, 2008. 1–15. [doi: 10.1007/978-3-540-89704-0_1]

[17] Del Vescovo C, Parsia B, Sattler U, Schneider T. The modular structure of an ontology: Atomic decomposition. In: Proc. of the 22nd Int'l Joint Conf. on Artificial Intelligence. 2011. 2232–2237. [doi: 10.5591/978-1-57735-516-8/IJCAI11-372]

[18] Grau BC, Halaschek-Wiener C. Incremental classification of description logics ontologies. Journal of Automated Reasoning, 2010,44:337–369. [doi: 10.1007/s10817-009-9159-0]

[19] Ji Q, Gao ZQ, Huang ZS, Zhu M. An efficient approach to debugging ontologies based on patterns. In: Proc. of the Joint Int'l Semantic Technology Conf. Heidelberg: Springer-Verlag, 2011. 425–433. [doi: 10.1007/978-3-642-29923-0_33]

[20] Horridge M. Justification based explanation in ontologies [Ph.D. Thesis]. Manchester: University of Manchester, 2011.

[21] Shchekotykhin K, Friedrich G, Dietmar J. On computing minimal conflicts for ontology debugging. In: Proc. of the ECAI 2008 Workshop on Model-Based Systems. 2008. 7–11.

[22] Ji Q, Gao ZQ, Huang ZS, Zhu M. Measuring effectiveness of ontology debugging systems. Knowledge-Based Systems, 2014,71: 169–186. [doi: 10.1016/j.knosys.2014.07.023]

[23] Lam SC, Pan JZ, Sleeman D, Vasconcelos W. A fine-grained approach to resolving unsatisfiable ontologies. Journal on Data Semantics X, 2800,62–95. [doi: 10.1007/978-3-540-77688-8_3]

[24] Du JF. Qi GL, Fu XF. A practical fine-grained approach to resolving incoherent owl 2 DL terminologies. In: Proc. of the 23rd ACM-CIKM. 2014. 919–928. [doi: 10.1145/2661829.2662046]

[25] Fu XF, Qi GL, Zhang Y. A graph-based approach for calculating minimal unsatisfiability-preserving subsets of ontology in DL-Lite. Chinese Journal of Electronics, 2016,44(9):2040–2045 (in Chinese with English abstract). [doi:10.3969/j.issn.0372-2112.2016.09.002]

[26] Fu XF, Qi GL, Zhang Y, Zhou, Z. Graph-Based approaches to debugging and revision of terminologies in DL-Lite. Knowledge-Based Systems, 2016,100:1–12. [doi: 10.1016/j.knosys.2016.01.039]

[27] Qi GL, Wang Z, Wang KW, *et al*. Approximating model-based ABox revision in DL-Lite: Theory and practice. In: Proc. of the 29th AAAI. AI Access Foundation. 2015. 254–260.

[28] Gao SB, Qi GL, Wang HF. A new operator for ABox revision in DL-Lite. In: Proc. of the 26th AAAI Conf. on Artificial Intelligence. El Segundo: AI Access Foundation, 2012. 2423–2424.

[29] Zhuang ZQ, Wang Z, Wang KW, Qi GL. Contraction and revision over DL-Lite TBoxes. In: Proc. of the 28th AAAI Conf. on Artificial Intelligence. El Segundo: AI Access Foundation, 2014. 1149–1155.

[30] Parsia B, Sirin E, Kalyanpur A. Debugging owl ontologies. In: Proc of the WWW-2005. 2005,1:633–640.

[31] Krötzsch M, Simancik F, Horrocks I. A description logic primer. Perspectives on Ontology Learning, 2014,18:3–20.

附中文参考文献:

[2] 李璞,蒋运承,王驹.基于保守扩充理论的模块化本体重用.软件学报,2016,27(11):2777–2795. http://www.jos.org.cn/1000-9825/4920.htm [doi: 10.13328/j.cnki.jos.004920]

[4] 欧阳丹彤,瞿剑峰,叶育鑫.关系抽取中基于本体的远监督样本扩充.软件学报,2014,25(9):2088–2101. http://www.jos.org.cn/1000-9825/4638.htm [doi: 10.13328/j.cnki.jos.004638]

[7] 欧阳丹彤,苏静,叶育鑫,崔仙姬.基于概念 R-MUPS 的本体调试方法.软件学报,2015,26(9):2231–2249. http://www.jos.org.cn/1000-9825/4735.htm [doi: 10.13328/j.cnki.jos.004735]

[25] 付雪峰,漆桂林,张勇.一种基于图的 DL-Lite 本体最小不可满足保持子集的计算方法.电子学报,2016,44(9):2040–2045.

张瑜(1982－),男,河南信阳人,博士生,主要研究领域为语义 Web,自动推理.

叶育鑫(1981－),男,博士,副教授,CCF 专业会员,主要研究领域为语义 Web,自动推理.

欧阳丹彤(1968－),女,博士,教授,博士生导师,CCF 高级会员,主要研究领域为模型诊断,语义 Web,自动推理.