一种约束增强的RDFS本体的模式验证方法

赵晓非, 柴争义, 袁超, 张振

引用本文

赵晓非, 柴争义, 袁超, 张振. 一种约束增强的RDFS本体的模式验证方法[J]. 计算机科学, 2024, 51(7): 362-372.

ZHAO Xiaofei, CHAI Zhengyi, YUAN Chao, ZHANG Zhen. Schema Validation Approach for Constraint-enhanced RDFS Ontology [J]. Computer Science, 2024, 51(7): 362-372.

计算机科学 COMPUTER SCIENCE
http://www.jsjkx.com
DOI:10.11896/jsjkx.230800034

一种约束增强的 RDFS 本体的模式验证方法

赵晓非[1]　柴争义[1]　袁　超[2]　张　振[2]
1 天津工业大学计算机科学与技术学院　天津 300387
2 天津工业大学软件学院　天津 300387

摘　要　约束增强的 RDFS(RDFS$_{(c)}$)本体克服了 RDFS 描述约束能力欠缺的缺点,然而约束机制的引入给本体验证问题带来了挑战。文中提出了一种面向 RDFS$_{(c)}$ 本体的、支持可判定性推断的模式验证方法。该方法对约束之间的依赖关系进行解析。首先,将 RDFS$_{(c)}$ 模式转化为一阶谓词逻辑表达式,而将规约的检测问题转化为表达式的可满足性检测问题;在此基础上,建立反映约束之间的修复-违背关系的约束依赖图,并对其进行必要的约简;接着,通过识别图中的有限环路来推导模式验证任务的可判定性;最后,通过约束依赖关系之上的推理进行模式的验证。该方法有两方面特点:一方面,通过到一阶谓词逻辑表达式的转换以及基于相应的一阶约束的依赖关系的推理,所提出的方法具有强适用性,特别是约束依赖性解析可以最大程度地减少回溯的次数,从而确保了验证过程的高效性;另一方面,由于独立于任何特定的约束建模语言,其也是一种分析 RDFS$_{(c)}$ 模式验证任务的可判定性的通用方法。

关键词:RDFS;本体验证;约束依赖性;可判定性
中图分类号　TP182

Schema Validation Approach for Constraint-enhanced RDFS Ontology

ZHAO Xiaofei[1],CHAI Zhengyi[1],YUAN Chao[2] and ZHANG Zhen[2]
1 School of Computer Science and Technology,Tiangong University,Tianjin 300387,China
2 School of Software Engineering,Tiangong University,Tianjin 300387,China

Abstract　The constraint-enhanced RDFS(RDFS$_{(c)}$) ontology overcomes the deficiency of RDFS's ability to describe constraints. However,the introduction of constraints brings challenges to ontology validation. This paper proposes a decidable schema validation approach for RDFS$_{(c)}$ ontology. This approach is based on analyzing the dependency between constraints. First,the RDFS$_{(c)}$ schema is transformed into the first-order predicate logic expressions,and the checking tasks of the characteristics are transformed into the satisfiability checking of the first-order expressions. On this basis,a constraint dependence graph reflecting the mend-violation relationships between constraints is established and made necessary reductions,we then derive the decidability of the schema validation tasks by identifying the finite loops in the graph,and finally validate the schema by reasoning on the constraint dependency. The contribution of this paper is twofold. On the one hand,through the transformation to the first-order expressions and the reasoning based on the dependency of the corresponding first-order constraints,the proposed approach has strong applicability,especially the constraint dependency analysis can minimize the number of backtracking,thereby ensuring the efficiency of the validation process;on the other hand,because it is independent of any specific constraint modeling language,this approach is also a general solution for analyzing the decidability of the RDFS$_{(c)}$ schema validation tasks.

Keywords　RDFS,Ontology validation,Constraint dependency,Decidability

1　引言

作为 W3C(World Wide Web Consortium)提出的对 Web 环境中的信息资源进行统一描述的语义模型,RDFS(Resource Description Framework Schema)[1]使得通过一系列有确定语义的词汇来描述概念之间的层次结构及概念和属性的语义成为可能,因而成为构建语义 Web 本体的重要基础。

确保本体的正确性,是基于本体的建模过程中避免错误传播的关键。为了确保 RDFS 本体的正确性所执行的推理,通常被称为 RDFS 本体验证[2-3]。模式验证是本体验证的

到稿日期:2023-08-07　返修日期:2024-01-14
基金项目:国家自然科学基金(62172298,61972456);江苏省计算机信息处理技术重点实验室开放基金(KJS1737)
This work was supported by the National Natural Science Foundation of China(62172298,61972456) and Open Foundation of Jiangsu Provincial Key Laboratory for Computer Information Processing Technology(KJS1737).
通信作者:赵晓非(zhaoxiaofei1978@hotmail.com)

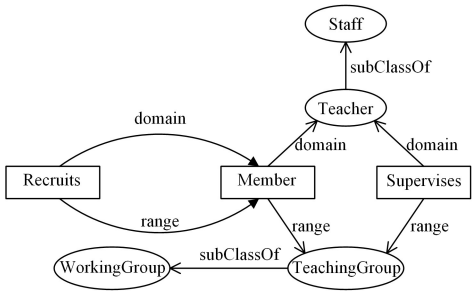
先决条件，该过程主要检测本体模式是否满足指定规约的集合。例如，一个典型的规约是类的可满足性。如果存在一个满足所有约束且包含至少一个类 C 实例的 RDFS 例示，则称类 C 是可满足的。RDFS 模式验证涉及的主要规约还包括属性的可满足性以及约束的非冗余性。

由于 RDFS 描述约束能力的欠缺，近年来在 RDFS 模式中引入非图形化的约束机制已成为一个重要的发展方向[4-6]。比较典型的工作包括：Nandana[7] 采用数据驱动的技术，利用数据简档作为 RDF 约束的载体，这些约束可以组合成 RDFShape，并用于规约的验证；Chmurovic 等[8] 针对形状约束语言 SHACL(Shapes Constraint Language)定义的 RDF 约束进行研究，并将 SHACL 约束和 Datalog 规则结合到一起，来检测推理规则在模式的图实例上的应用可能会违反哪些约束；Garcia 等[9]将形状表达式语言 ShEx(Shape Expressions)定义的约束纳入研究范围，所提出的方法可以根据 RDF 数据集的结构特征半自动地构建约束，并支持通过集成验证器对模式的规约进行验证；Fernandez 等[10] 采用一种支持否定和递归的表达式机制来建模 RDF 约束，该机制还包含代数分组运算符、选择运算符、基数约束和布尔运算符，他们同时给出了其语法及形式语义，并针对上述机制研究模式验证算法。在工具实现方面，Robaldo 等开发了基于回答集编程(Answer Set Programming，ASP)的验证工具 DLV2[11]。该工具能够解析 RDF 文件中包含的三元组并自动将其转换为 ASP 事实，而将待验证的规约形式化为一阶逻辑规则。Prud 等提出了 FHIR[12]将其并应用于医疗保健领域。该工具被用以验证 RDF 数据是否符合 FHIR 规范中提供的 ShEx 模式。为了在持续集成(Continuous Integration，CI)环境中进行 RDF 数据验证，从而确保数据在每次迭代后仍与其模式一致，Publio 等开发了 Ontolo-CI[13]，该工具可以在持续更新的本体文件上运行预定义或自定义的 ShEx 验证。

引入了非图形约束的 RDFS 本体通常被称为约束增强的 RDFS 本体，即 $RDFS_{(c)}$ 本体[4]。由于现实世界中的约束条件具有多样性及复杂性，非图形约束的引入使得 $RDFS_{(c)}$ 模式的验证问题变得更加复杂，甚至可能导致该任务变为不可判定的，因此尽管已有上述研究成果被提出，如何推导模式验证问题的可判定性并高效地对模式进行验证仍未得到很好的解决，目前已成为 $RDFS_{(c)}$ 本体建模领域的研究热点之一。

针对已有的技术无法识别验证任务的可判定性以及效率不高的问题，本文提出了一种基于约束依赖性解析的方法。为了把 $RDFS_{(c)}$ 模式及待检测的规约统一于一种语义框架下，首先将二者转化为一阶谓词逻辑表达式，在此基础上通过分析肯定文字和否定文字对逻辑表达式的不同影响，研究了约束之间的修复-违背关系，并据此建立反映此关系的约束依赖图。为了剔除对结果无影响的非依赖弧，我们给出了约束依赖图的约简策略，然后通过识别图中的有限环路来推导出问题的可判定性。对于每一种约简策略和识别策略，都给出了严格的形式化证明。

下文将通过两个例子来阐述所提出的方法。例 1(如图 1 所示)中的 $RDFS_{(c)}$ 模式描述了教师(Teacher)与教学团队(TeachingGroup)之间的关系。教师可以是教学团队的督导(用一般对象属性 Supervises 表示)，也可以是存在招募关系的教学团队成员(用类属对象属性 Member 表示)。附加的非图形约束分别限制：每个教学团队有且只能有一个督导(约束 1)；每个教学团队至少拥有一名成员(约束 2)；任何教学团队的成员只能被至多一名成员所招募(约束 3)；作为教学团队的督导的教师不能是该团队的成员之一(约束 4)；教学团队的成员不能自己招募自己(约束 5)；每个教学团队中至少有一名成员是被本团队的其他成员所招募来的(约束 6)。为了不失一般性，非图形约束均以一阶谓词逻辑公式表示。



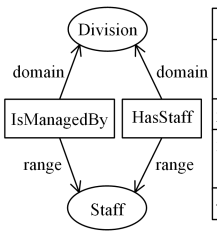
Additional constraints:
1. TeachingGroup(G)→(∃T)Supervises(T,G)∧Teacher(T) Supervises(T_1,G)∧Supervises(T_2,G)→T_1=T_2
2. TeachingGroup(G)→(∃T)(∃M)Member(M,T,G)∧Teacher(T)
3. Recruits(M_1,N)∧Recruits(M_2,N)→M_1=M_2
5. Member(M,T_1,G_1)∧Member(N,T_2,G_2)∧Recruits(M,N)→T_1<>T_2
4. Supervises(T,G)→¬(∃M)Member(M,T,G)
6. TeachingGroup(G)→(∃M)(∃N)(∃T_1)(∃T_2)Member(M,T_1,G)∧ Member(N,T_2,G)∧Recruits(M,N)∧Teacher(T_1)∧Teacher(T_2)

图 1 描述教师和教学团队之间关系的 $RDFS_{(c)}$ 模式

Fig. 1 $RDFS_{(c)}$ Schema describes the relationship between teacher and teaching group

例 2(如图 2 所示)中的 $RDFS_{(c)}$ 模式描述了职员(Staff)和部门(Division)之间的关系。Staff 既可以是普通职员，也可以是部门的管理者，因此通过 HasStaff 和 IsManagedBy 两种对象属性关联到 Division。附加的 4 个非图形约束分别用以限定：Staff 和 Division 通过 IsManagedBy 相关联时，实例数量的限制(约束 1 和约束 2)；Staff 和 Division 通过 HasStaff 相关联时，实例数量的限制(约束 3 和约束 4)。

运用本文的方法，我们发现例 1 对应的约束依赖图中存在的所有环路都属于有限环路。由于检测类和属性的可满足性、约束的非冗余性的任务均可转化为一阶谓词逻辑表达式的可满足性检测，因此可以推出在该 $RDFS_{(c)}$ 模式中的上述验证任务均是可判定的。在此基础上运用所提出的检测算法，实验结果表明所提方法能够高效、准确地对该模式进行验证。相反，例 2 对应的约束依赖图中存在 2 个环路，其中之一不是有限环路，因此在该 $RDFS_{(c)}$ 模式中的上述验证任务是不可判定的。实际上，例 2 中的模式对应于一个经典的 E-R 模型，Lenzerini 等[14] 已经证明了在该 E-R 模型中实体和关系的可满足性推理任务是不可判定的，在此我们借用了该例来说明本方法的过程及有效性。



Additional constraints:
1. Division(D)→(∃S)IsManagedBy(D,S)∧Staff(S) IsManagedBy(D,S_1)∧IsManagedBy(D,S_2)→S_1=S_2
2.Staff(S)→(∃D)Division(D)∧IsManagedBy(D,S)
3. Division(D)→(∃S1)(∃S_2)Staff(S_1)∧Staff(S_2)∧ HasStaff(D,S_1)∧HasStaff(D,S_2)∧S_1<>S_2
4. HasStaff(D_1,S)∧HasStaff(D_2,S)→D_1=D_2

图 2 描述职员和部门之间关系的 $RDFS_{(c)}$ 模式

Fig. 2 $RDFS_{(c)}$ Schema describes the relationship between staff and division

2 约束依赖性解析的逻辑基础及本方法的整体思路

2.1 约束依赖性解析的逻辑基础

限于篇幅，我们仅对与本文关系密切的逻辑机制进行简要介绍。

变元和常量统称为项，记为 t。原子 $P(t_1,\cdots,t_n)$ 是作用于 n 个项 $t_1,\cdots,t_n$ 的谓词，简记为 $P(\bar{t})$。如果一个原子中的每个 t_i 都是常量，则称该原子为闭原子。原子 $P(\bar{t})$ 及其否定原子 $\neg P(\bar{t})$ 统称为文字，记为 ℓ。

一个标准子句 S 是形如 $\ell_1\wedge\cdots\wedge\ell_m\rightarrow\mathcal{H}(m\geqslant 0)$ 的表达式，其中 $\mathcal{H}$ 为原子，每个 ℓ_i 为文字。$\mathcal{H}$ 和 $\ell_1\wedge\cdots\wedge\ell_m$ 分别称为标准子句的头(记为 head(S))和体(记为 body(S))。根据头和体的不同取值，标准子句可进一步分为 3 种形式。

(1)事实：事实是形如 $\rightarrow P(\bar{A})$ 的标准子句，也可简写为 $P(\bar{A})$，其中 $P(\bar{A})$ 为闭原子。例如 Teacher(Lucy)和 Mother(Lucy,Linda)是分别陈述 Lucy 是教师以及 Lucy 是 Linda 的母亲的事实。

(2)演绎规则：演绎规则是形如 $\ell_1\wedge\cdots\wedge\ell_m\rightarrow P(\bar{t})(m\geqslant 1)$ 的标准子句，其中 P 是演绎规则所定义的谓词名。一个谓词名 P 的定义是将 P 作为头的所有演绎规则的集合。例如下述 2 条演绎规则定义了谓词名 Aunt：

$Sister(x,y)\wedge Father(y,z)\rightarrow Aunt(x,z)$

$Sister(x,y)\wedge Mother(y,z)\rightarrow Aunt(x,z)$

(3)条件：条件是形如 $\ell_1\wedge\cdots\wedge\ell_m\rightarrow\perp(m\geqslant 1)$ 的标准子句，表示永远不能被满足的情形。例如，条件 $Aunt(x,y)\wedge Sister(x,y)\rightarrow\perp$ 表示 $Aunt(x,y)$ 和 $Sister(x,y)$ 永远不能同时被满足。任何形式的一阶逻辑表达式都可以在有限步骤内转换为与之等价的标准子句并进一步转换为与之等价的条件，具体步骤参见文献[15]。

项、文字以及由它们构成的语法结构(事实、演绎规则等)，统称为表达式。若 ε 是表达式，则 $var(\varepsilon)$ 表示 ε 中出现的所有变元的集合。一个逻辑模式 $\mathcal{LS}$ 是一个二元组($\mathcal{DR}$，$\mathcal{NC}$)，其中 $\mathcal{DR}$ 是演绎规则的有限集合，而 $\mathcal{NC}$ 是条件的有限集合，前者用于定义 $\mathcal{LS}$ 中的规则，后者用于定义 $\mathcal{LS}$ 中的约束。在 $\mathcal{DR}$ 和 $\mathcal{NC}$ 的体中出现的谓词分为外延谓词和内涵谓词 2 种，前者是事实中直接给定的关系，而后者是 $\mathcal{DR}$ 中的演绎规则所定义的关系。给定逻辑模式 $\mathcal{LS}$，$\mathcal{LS}$ 的例示 I 是一个二元组($\mathcal{E}$，$\mathcal{LS}$)，其中 $\mathcal{E}$ 是外延谓词的事实的有限集合。$\mathcal{DR}(\mathcal{E})$ 表示根据 $\mathcal{DR}$ 中的演绎规则所推导出的事实(即内涵谓词的事实)与 $\mathcal{E}$(即外延谓词的事实)的并集。例如，给定 $\mathcal{E}=\{Sister(Rose,Tom),Sister(Rose,Lucy),Father(Tom,Betty),Mother(Lucy,Linda),Sister(Rose,Linda)\}$，令 $\mathcal{DR}$ 是前述的定义谓词 $Aunt$ 的 2 条演绎规则构成的集合，经过推理可得 $\mathcal{DR}(\mathcal{E})=\mathcal{E}\cup\{Aunt(Rose,Betty),Aunt(Rose,Linda)\}$。

置换 θ 是形如 $\{x_1/t_1,\cdots,x_n/t_n\}$ 的一个集合，其中每个变元 x_i 互不相同且每个项 $t_i\neq x_i$。所有 x_i 构成的集合被称为置换 θ 的域，记为 $domain(\theta)$。如果每个 t_i 均为常量，则称置换 θ 为闭置换。应用置换 θ 将表达式 $\mathcal{E}$ 中的每个 x_i 同步替换为相应的 t_i 所得到的表达式，记为 $\varepsilon\theta$。给定例示 I 及条件 $\ell_1\wedge\cdots\wedge\ell_m\rightarrow\perp(m\geqslant 1)$，如果存在闭置换 θ 使得 $I\vDash(\ell_1\wedge\cdots\wedge\ell_m)\theta$，则称 I 违背该条件。换句话说，例示 I 违背条件 $\ell_1\wedge\cdots\wedge\ell_m\rightarrow\perp(m\geqslant 1)$ 当且仅当 $\{\ell_1\theta,\cdots,\ell_m\theta\}\subseteq\mathcal{DR}(\mathcal{E})$。例如，由于存在闭置换 $\theta=\{x/Rose,y/Linda\}$ 使得 $\{Aunt(x,y)\theta,Sister(x,y)\theta\}=\{Aunt(Rose,Linda),Sister(Rose,Linda)\}\subseteq\mathcal{DR}(\mathcal{E})$，因此前述的例示违背了条件 $Aunt(x,y)\wedge Sister(x,y)\rightarrow\perp$。如果例示 I 不违背 $\mathcal{NC}$ 中的任何一个条件，则称 I 关于 $\mathcal{NC}$ 是一致的。

2.2 本方法的整体思路

如前所述，给定 $RDFS_{(c)}$ 模式和待检测的规约(简记为 STC(Statute to be Checked))，模式验证过程是检测该模式是否满足 STC 的过程。为了推导该过程的可判定性并实现验证，我们借助逻辑机制对 $RDFS_{(c)}$ 模式进行形式化描述，将其转化为一组一阶约束。进一步，将模式中的类、数据属性和对象属性转化为一阶谓词，而模式中的图形约束和非图形约束转化为一组条件表达式(即一组一阶约束)。STC 的检测问题则转化为一阶表达式的可满足性检测问题。例如，检测 TeachingGroup 可满足性的任务转化为 $TeachingGroup(g)$ 的可满足性检测，而检测是否存在拥有两个督导的教学团队的任务则转化为 $Supervises(t_1,g)\wedge Supervises(t_2,g)\wedge t_1\langle\rangle t_2$ 的可满足性检测。

由于 STC 所对应的一阶表达式的可满足性检测过程是搜索既满足约束也满足 STC 的例示的过程，因此检测 STC 是否是可满足的等价于判断是否可以修复一个满足 STC 的例示。假定存在某个满足 STC 的例示 I，如果 I 违背某个约束，则通过 I 是无法判断 STC 的可满足性的，然而在满足 STC 的前提下可以通过向 I 中添加有限数量的新实例来尝试修复被违背的约束。新实例的引入有可能导致其他约束被违背，因此该过程是一个迭代的过程，反复添加新实例修复被违背的约束，直至得到一个与所有约束集合一致的例示或者存在被违背的约束且无法再进行修复为止。前者意味着 STC 是可满足的，而后者意味着 STC 是不可满足的。

通过这种分析不难看出，如果上述过程可以在有限步骤内结束，则 STC 的检测问题是可判定的，否则该问题是不可判定的，因此影响该问题可判定性的约束无外乎两种情况：(1)试图满足 STC(或修复随后被违背的约束)时会被违背的约束；(2)当被修复时将违背其他约束的约束。以图 1 为例，假定要找到一个例示以证明 TeachingGroup 是可满足的。仅包含一个 TeachingGroup 的实例的例示满足 TeachingGroup 但却违背了每个教学团队至少拥有一名成员的约束。为了

修复该约束，需要向例示中添加与 TeachingGroup 相关联的一个 Teacher 的实例，然而该实例的添加进一步违背了另一条约束，即每个教学团队中至少有一名成员是被本团队的其他成员所招募来的。因此第一个约束会影响 TeachingGroup 的可满足性的判定，因为试图满足 STC 时它会被违背并且当修复它时也将违背其他约束。由于当第二个约束被违背时总是可以通过添加 Teacher 的一个新的实例并建立二者之间的招募关系来进行修复，而这种修复不会违背其他任何约束，因此第二个约束不会影响 TeachingGroup 可满足性的判定。我们将建立约束依赖图，以便反映约束之间的这种关系。

总体而言，本方法验证 $RDFS_{(c)}$ 模式是否满足指定的规约分 4 步进行：(1)将 $RDFS_{(c)}$ 模式及 STC 形式化为一阶谓词逻辑表达式；(2)建立约束依赖图并对其进行约简；(3)识别有限环路从而确定任务的可判定性；(4)执行可满足性检测算法对模式进行验证。第 3－6 节将分别对这 4 步进行详细介绍。

3 $RDFS_{(c)}$ 模式和规约的形式化

3.1 $RDFS_{(c)}$ 模式的形式化

我们对模式中的类、一般对象属性、类属对象属性、数据属性、图形约束和非图形约束分别进行形式化，具体的方法如表 1 所列。

表 1 $RDFS_{(c)}$ 模式元素的形式化

Table 1 Formalization for elements of a $RDFS_{(c)}$ schema

	$RDFS_{(c)}$ 模式元素	逻辑表达式
基本元素	类 C	$C(c)$
	类 C_1 和 C_2 之间的一般对象属性 R	$R(c_1,c_2)$
	类 C_1 和 C_2 之间的类属对象属性 R	$R(r,c_1,c_2)$
	类 C 的数据属性 A	$CA(c,a)$
图形约束	类层次：类 C_1 是类 C_2 的子类	$C_1(c)\wedge\neg C_2(c)\rightarrow\perp$
	类 C_1 和 C_2 之间的一般[类属]对象属性 R 的完整性	$R([r,]c_1,c_2)\neg C_1(c_1)\rightarrow\perp$ $R([r,]c_1,c_2)\neg C_2(c_2)\rightarrow\perp$
	类 C_1 和 C_2 之间的类属对象属性 R 的实例唯一性	$R(r_1,c_1,c_2)\wedge R(r_2,c_1,c_2)\wedge r_1\langle\rangle r_2\rightarrow\perp$
	类 C 的数据属性 A 的完整性	$CA(c,a)\neg C(c)\rightarrow\perp$
非图形约束	类 C_1 和 C_2 之间的一般[类属]对象属性 R 在 C_1 方的下限 $lower$ 约束($lower$ 为正整数)	$C_2(c_2)\neg lowerR(c_2)\rightarrow\perp$ $R([r,]c_{11},c_2)\wedge R([r,]c_{12},c_2)\wedge\cdots\wedge R([r,]c_{1lower-1},c_2)\wedge R([r,]c_{1lower},c_2)\wedge c_{11}\langle\rangle c_{12}\wedge\cdots\wedge c_{1lower-1}\langle\rangle c_{1lower}\rightarrow lowerR(c_2)$
	类 C_1 和 C_2 之间的一般[类属]对象属性 R 在 C_1 方的上限 $upper$ 约束($upper$ 为正整数)	$R([r,]c_{11},c_2)\wedge R([r,]c_{12},c_2)\wedge\cdots\wedge R([r,]c_{1upper},c_2)\wedge R([r,]c_{1upper+1},c_2)\wedge c_{11}\langle\rangle c_{12}\wedge\cdots\wedge c_{11}\langle\rangle c_{1upper+1}\wedge\cdots\wedge c_{1upper}\langle\rangle c_{1upper+1}\rightarrow\perp$
	类 C_1 和 C_2 之间的一般[类属]对象属性 R 在 C_2 方的下限 $lower$ 约束($lower$ 为正整数)	$C_1(c_1)\neg lowerR(c_1)\rightarrow\perp$ $R([r,]c_1,c_{21})\wedge R([r,]c_1,c_{22})\wedge\cdots\wedge R([r,]c_1,c_{2lower-1})\wedge R([r,]c_1,c_{2lower})\wedge c_{21}\langle\rangle c_{22}\wedge\cdots\wedge c_{2lower-1}\langle\rangle c_{2lower}\rightarrow lowerR(c_1)$
	类 C_1 和 C_2 之间的一般[类属]对象属性 R 在 C_2 方的上限 $upper$ 约束($upper$ 为正整数)	$R([r,]c_1,c_{21})\wedge R([r,]c_1,c_{22})\wedge\cdots\wedge R([r,]c_1,c_{2upper})\wedge R([r,]c_1,c_{2upper+1})\wedge c_{21}\langle\rangle c_{22}\wedge\cdots\wedge c_{21}\langle\rangle c_{2upper+1}\wedge\cdots\wedge c_{2upper}\langle\rangle c_{2upper+1}\rightarrow\perp$

对于模式中的基本元素，我们把每个类形式化为一个一元谓词，每个一般对象属性和每个数据属性分别形式化为一个二元谓词，而每个类属对象属性则形式化为一个三元谓词，所有谓词中的参数均为变元。例如，类 *Teacher* 被形式化为 $Teacher(t)$，其中 t 是 *Teacher* 的实例变元。类 *Teacher* 和 *TeachingGroup* 之间的一般对象属性 *Supervises* 被形式化为 $Supervises(t,g)$，其中 t 和 g 分别是 *Teacher* 和 *TeachingGroup* 的实例变元。类 *Teacher* 和 *TeachingGroup* 之间的类属对象属性 *Member* 被形式化为 $Member(m,t,g)$，其中 m 是 *Member* 的实例变元。

在此基础上，每个图形约束被形式化为表示一阶约束的条件表达式。例如，类 *Teacher* 和 *Staff* 之间的类层次被形式化为 $Teacher(t)\wedge\neg Staff(t)\rightarrow\perp$，该条件表达式精确地指定了所有 *Teacher* 的实例必须也是 *Staff* 的实例。一般对象属性 *Supervises* 的完整性被形式化为 2 个条件表达式：$Supervises(t,g)\wedge\neg TeachingGroup(g)\rightarrow\perp$ 和 $Supervises(t,g)\wedge\neg Teacher(t)\rightarrow\perp$。前者指定 *Supervises* 的值域是 *Teaching-Group*，而后者指定 *Supervises* 的定义域是 *Teacher*。类属对象属性 *Member* 的实例唯一性被形式化为 $Member(m_1,t,g)\wedge Member(m_2,t,g)\wedge m_1\langle\rangle m_2\rightarrow\perp$，该条件表达式精确地指定了被同一对 *Teacher* 和 *TeachingGroup* 的实例所定义的 *Member* 的实例必须是唯一的。

对于每个非图形约束，我们首先建立表示该约束的一阶逻辑表达式，接着对其进行标准子句化(参见文献[15])，将其转化为等价的标准子句，并进一步转化为等价的条件表达式。通过这种方式，每个非图形约束被形式化为表示一阶约束的条件表达式，从而融入与图形约束相同的形式化体系。为了不失一般性，表 1 中仅列出了下限约束和上限约束的形式化方法，其他非图形约束均可按照这种步骤进行形式化。需要特别指出的是，此过程中可能会有辅助谓词被引入。例如，约束“每个教学团队至少有一名督导”将被形式化为 $TeachingGroup(g)\wedge\neg oneSuperviser(g)\rightarrow\perp$ 和 $Supervises(t,g)\rightarrow oneSuperviser(g)$，其中的 $oneSuperviser(g)$ 即为向条件表达式转化过程中引入的辅助谓词。直观上看，$Supervises(t,g)\rightarrow oneSuperviser(g)$ 用以表示如果存在 *Teacher* 的实例 t 对 *TeachingGroup* 的实例 g 进行监督指导，则 g 达到下限要求；而

$TeachingGroup(g) \wedge \neg oneSuperviser(g) \rightarrow \perp$ 则表示如果 g 是 $TeachingGroup$ 的实例而 g 没有达到下限要求，则约束被违背。

应用表 1 中的方法对图 1 中的 $RDFS_{(c)}$ 模式进行形式化，得到的结果如表 2 所列。

表 2 图 1RDFS$_{(c)}$ 模式对应的一阶约束

Table 2 First-order constraints for the RDFS$_{(c)}$ schema in Fig. 1

图形约束	类层次 $Con_1: Teacher(t) \wedge \neg Staff(t) \rightarrow \perp$ $Con_2: TeachingGroup(g) \wedge \neg WorkingGroup(g) \rightarrow \perp$ 对象属性的完整性 $Con_3: Supervises(t,g) \wedge \neg TeachingGroup(g) \rightarrow \perp$ $Con_4: Supervises(t,g) \wedge \neg Teacher(t) \rightarrow \perp$ $Con_5: Member(m,t,g) \wedge \neg TeachingGroup(g) \rightarrow \perp$ $Con_6: Member(m,t,g) \wedge \neg Teacher(t) \rightarrow \perp$ $Con_7: Recruits(m_1,m_2) \wedge \neg beMember(m_1) \rightarrow \perp$ $Member(m,t,g) \rightarrow beMember(m)$ $Con_8: Recruits(m_1,m_2) \wedge \neg beMember(m_2) \rightarrow \perp$ 类属对象属性的实例唯一性 $Con_9: Member(m_1,t,g) \wedge Member(m_2,t,g) \wedge m_1 \langle\rangle m_2 \rightarrow \perp$
非图形约束	约束 1 $Con_{10}: TeachingGroup(g) \wedge \neg oneSuperviser(g) \rightarrow \perp$ $Supervises(t,g) \rightarrow oneSuperviser(g)$ $Con_{11}: Supervises(t_1,g) \wedge Supervises(t_2,g) \wedge t_1 \langle\rangle t_2 \rightarrow \perp$ 约束 2 $Con_{12}: TeachingGroup(g) \wedge \neg oneMember(g) \rightarrow \perp$ $Member(m,t,g) \rightarrow oneMember(g)$ 约束 3 $Con_{13}: Recruits(m_1,n) \wedge Recruits(m_2,n) \wedge m_1 \langle\rangle m_2 \rightarrow \perp$ 约束 4 $Con_{14}: Supervises(t,g) \wedge Member(m,t,g) \rightarrow \perp$ 约束 5 $Con_{15}: Member(m,t_1,g_1) \wedge Member(n,t_2,g_2) \wedge Recruits(m,n) \wedge t_1 = t_2 \rightarrow \perp$ 约束 6 $Con_{16}: TeachingGroup(g) \wedge \neg onebeRecruited(g) \rightarrow \perp$ $Member(m,t_1,g) \wedge Member(n,t_2,g) \wedge Recruits(m,n) \wedge t_1 \langle\rangle t_2 \rightarrow onebeRecruited(g)$

3.2 RDFS$_{(c)}$ 规约的形式化

如前所述，需要将 STC 形式化为一阶表达式，从而将 STC 的检测问题转化为其对应的一阶表达式的可满足性检测问题。如果表达式是可满足的，则说明存在既满足所有约束也满足该 STC 的例示，即该 STC 成立。

类 C 的可满足性检测将直接转化为 $C(c)$ 的可满足性检测，而对象属性 R 的可满足性检测则直接转化为 $R([r,] c_1, c_2)$ 的可满足性检测。下面讨论约束的非冗余性检测。一个约束是冗余的，当且仅当该约束所禁止的例示已被其他约束所避免。换句话说，如果删除某约束后，存在满足其他所有约束但违背该约束的例示，则该约束是非冗余的。尽管冗余的约束不影响 RDFS$_{(c)}$ 本体的正确性，它却使得本体的修改更加困难并且容易导致约束之间的冲突。此外，冗余的约束也会降低本体推理的效率。检测约束非冗余性的任务将转化为该约束所对应的条件表达式的体的可满足性检测。例如，检测例 1 中约束 4 的非冗余性将被转化为 $Supervises(t,g) \wedge Member(m,t,g)$ 的可满足性检测。如果上式是不可满足的，则说明不可能存在这样的例示，其中教师 t 既是教学团队 g 的督导也是 g 的成员之一，也就是说，删除约束 4 后，它所禁止的例示已被其他约束所避免，因此可知约束 4 是冗余的。

4 RDFS$_{(c)}$ 模式的约束依赖性解析

4.1 约束依赖图的建立

通过 3.1 节所述的形式化，已将 RDFS$_{(c)}$ 模式中的所有约束转化为一阶条件表达式。由于每个条件表达式表示一种不能被满足的情形，因此当条件的体中的所有肯定文字中的原子为真（即在 $\mathcal{DR}(\mathcal{I})$ 中出现），而所有否定文字中的原子为假（即在 $\mathcal{DR}(\mathcal{I})$ 中不出现）时，该条件被违背。

定义 1 条件 Con 的体中的每个肯定文字 $P(\bar{t})$，称为 Con 的一个潜在违背。条件 Con 的所有潜在违背构成的集合被称为 Con 的潜在违背集，记为 $Pc(Con)$。

术语“潜在违背”用于表明并不是体中的肯他文字中的原子的出现必然导致 Con 的违背，而是“可能”导致 Con 的违背，仅当体中的其它肯定文字中的原子都出现且所有否定文字中的原子都不出现时会导致 Con 的违背。举例来说，当 $\mathcal{DR}(\mathcal{I})$ 中不存在事实 $TeachingGroup(B)$ 时，$Supervises(A,B)$ 的出现将导致 Con_3 的违背，因此 $Supervises(t,g)$ 是 Con_3 的一个潜在违背。由于 Con_3 的体中不包含其他肯定文字，因此 $Pc(Con_3)=\{Supervises(t,g)\}$。对于 Con_{14}，当 $\mathcal{DR}(\mathcal{I})$ 中包含事实 $Member(M,C,D)$ 时，$Supervises(C,D)$ 的出现将导致该约束的违背，反之亦如此，因此 $Pc(Con_{14})=\{Supervises(t,g), Member(m,t,g)\}$。

定义 2 对于条件 Con 的体中的每个否定文字 $\neg P(\bar{t})$：(1)如果 $P(\bar{t})$ 不是由演绎规则所定义的原子，则每个 $\{P(\bar{t})\}$ 是 Con 的一个修复；(2)如果 $P(\bar{t})$ 是由演绎规则所定义的原子，则定义 $P(\bar{t})$ 的每个演绎规则的体中出现的所有肯定文字的集合是 Con 的一个修复。条件 Con 的所有修复构成的集合称为 Con 的修复集，记为 $Re(Con)$。

对于条件 Con 的体中的每个否定文字 $\neg P(\bar{t})$，$P(\bar{t})$ 在 $\mathcal{DR}(\mathcal{I})$ 中出现意味着 $\neg P(\bar{t})$ 为假，进而避免了 Con 被违背，因此 $P(\bar{t})$ 的出现实际上修复了 Con。令 $Re_k(Con)$ 为 $Re(Con)$ 中的第 k 个修复，由于 Con 被违背时无论向例示中添加哪个 $Re_k(Con)$ 的实例都可以修复该 Con，因此每个 $Re_k(Con)$ 是一种修复 Con 的不同方法。举例来说，对于 Con_3，由于 $TeachingGroup(g)$ 不是由演绎规则所定义的原子，因此 $\{TeachingGroup(g)\}$ 是 Con_3 的一个修复，由于 Con_3 的体中不包含其他否定文字，因此 $Re(Con_3)=\{\{TeachingGroup(g)\}\}$。对于 Con_{16}，由于 $onebeRecruited(g)$ 是由演绎规则所定义的原子，因此 $\{Member(m,t_1,g), Member(n,t_2,g), Recruits(m,n)\}$ 是 Con_{16} 的一个修复。由于 Con_{16} 的体中不包含其他否定文字，因此 $Re(Con_{16})=\{\{Member(m,t_1,g), Member(n,t_2,g), Recruits(m,n)\}\}$。对于修复集为空的约束（比如 Con_{14}），这些约束一旦被违背，将永远无法被修复。

基于 RDFS$_{(c)}$ 模式对应的条件表达式的集合，下面将建立（为了使某个约束可满足而应用的）修复和（应用该修复可能导致的其他约束的）违背之间的相互影响关系。进一步，如果某个 Con 的一个修复与另一个 Con 的潜在违背集包含相同的原子，通过添加该原子的新实例来修复前一个 Con 可能导致后一个 Con 被违背，因此这两个 Con 之间存在修复-违背关系。我们通过建立约束依赖图来反映这种关系。

定义 3 一个约束依赖图 G 是一个有向图，其中每个顶点

对应于一个约束。令 i,j 表示约束的序号，k 表示修复的序号，从顶点 Con_i 到 Con_j 存在有向弧(标记为 $Re_k(Con_i)$)，当且仅当存在原子 $P(\bar{t})$ 和置换 θ 满足 $P(\bar{t})\in Re_k(Con_i)$ 并且 $P(\bar{t})\theta\in Pc(Con_j)$，其中 $Re_k(Con_i)\in Re(Con_i)$。

根据定义 3，如果约束 Con_i 的某个修复 $Re_k(Con_i)$ 与 $Pc(Con_j)$ 包含相同的原子，即 Con_j 可能由于 Con_i 的该修复而被违背，则建立从 Con_i 到 Con_j 的有向弧以表示这种关系，并将该弧标记为 $Re_k(Con_i)$。例如，Con_7 的第一个修复{$Member(m,t,g)$}与 Con_{14} 的潜在违背集{$Supervises(t,g)$，$Member(m,t,g)$}都包含原子 $Member(m,t,g)$，因此添加 $Member(m,t,g)$ 的一个实例来修复 Con_7 就会导致 Con_{14} 被违背(如果违背 Con_{14} 的其他条件都满足)，我们建立从 Con_7 到 Con_{14} 的有向弧并将其标记为{$Member(m,t,g)$}。按照定义 3，基于表 2 中的所有一阶约束建立起约束依赖图，如图 3 所示。

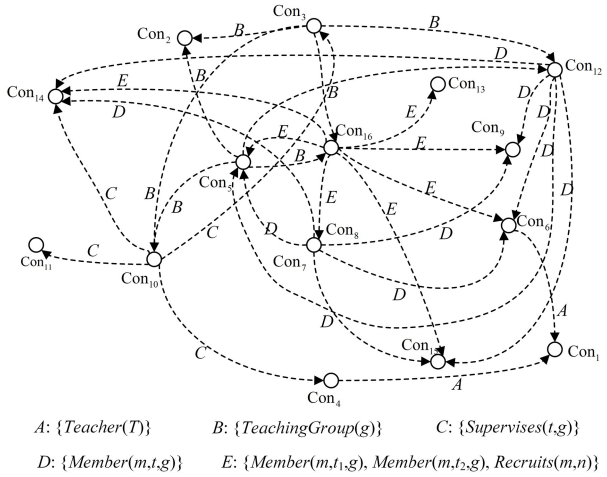


图 3 基于表 2 中的一阶约束建立的约束依赖图

Fig. 3 Constraint dependency graph established based on the first-order constraints in Table 2

4.2 约束依赖图的约简

通过深入分析约束依赖图可以发现，尽管某些顶点之间存在弧，但是弧的始端约束的修复并不会导致其终端约束的违背。由于将这些事实上不会导致违背发生的有向弧从约束依赖图中删除不会影响模式验证的结果，因此称这种可删除的弧为非依赖弧。例如对于图 3 中的 Con_3，向 $\mathcal{DR}(\mathcal{E})$ 中添加事实 Supervises(A,B)导致了 Con_3 被违背，其相应的修复 TeachingGroup(B)却永远不会导致 Con_{10} 的违背(由于 Con_{10} 的修复已存在于 $\mathcal{DR}(\mathcal{E})$ 中)。再比如 Con_{10} 的修复也永远不会导致 Con_{11} 的违背。由于二者分别描述每个教学团队至少和至多拥有一名督导，而上限约束并不会由于修复下限约束而被违背(因为如果添加了 $TeachingGroup(A)$ 而导致 Con_{10} 被违背，修复 Con_{10} 意味着之前 $TeachingGroup$ 的实例 A 并没有通过 $Supervises$ 关联到任何 $Teacher$ 的实例)，因此 Con_3 到 Con_{10}、Con_{10} 到 Con_{11} 的弧都是不影响模式验证结果的非依赖弧。

定义 4 给定约束依赖图 G，令 $Re_k(Con_i)$ 是从 Con_i 到 Con_j 的弧，其中 i,j 表示约束的序号，k 表示修复的序号，对于任意例示 $I=(\mathcal{E},\mathcal{LS})$，如果均存在置换 θ 使得 $\mathcal{DR}(\mathcal{E})\cup(Re_k(Con_i))\models\neg body(Con_j)\theta$，则称 $Re_k(Con_i)$ 为非依赖弧。

定义 5 如果变元 x 满足 $x\in var(Re_k(Con))$ 并且 $x\notin var(Pc(Con))$，其中 k 表示修复的序号，则称 x 是 $Re_k(Con)$ 的增量变元。$Re_k(Con)$ 的所有增量变元的集合记为 $addition(Re_k(Con))$。

下面给出删除非依赖弧的策略。以图 4 中的 2 个约束为例来进行说明。与前面类似，i 和 j 表示约束的序号，h 和 k 表示修复的序号。

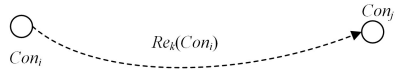


图 4 删除非依赖弧的用例

Fig. 4 Example of deleting non-dependent arc

定理 1 若存在置换 θ 及修复 $Re_h(Con_j)$ 满足 $Re_h(Con_j)\theta\subseteq Pc(Con_i)\cup Re_k(Con_i)$，其中 $domain(\theta)\subseteq addition(Re_h(Con_j))$，则弧 $Re_k(Con_i)$ 是非依赖弧。

证明：假设例示 $I\not\models Con_i$，通过向 I 中添加 $Re_k(Con_i)$ 的实例得到的例示为 I' 且 $I'\models Con_i$。根据 $I\not\models Con_i$ 可得 $I\models Pc(Con_i)\wedge I\not\models Re_k(Con_i)$。再考虑 I' 的建立过程，可知 $I'\models Pc(Con_i)\cup Re_k(Con_i)$。根据存在置换 θ 满足 $Re_h(Con_j)\theta\subseteq Pc(Con_i)\cup Re_k(Con_i)$ 且 $domain(\theta)\subseteq addition(Re_h(Con_j))$ 可得 $I'\models Re_h(Con_j)\theta$，也就是说 Con_j 的修复 $Re_h(Con_j)$ 已经成立，因此 $I'\models Con_j$。证毕。

直观地说，每当 $Pc(Con_i)$ 成立，通过添加 $Re_k(Con_i)$ 的实例进行修复时，$Re_h(Con_j)$ 总是成立，因为它已包含于 $Pc(Con_i)\cup Re_k(Con_i)$ 中，因此尽管 $Pc(Con_j)$ 可能由于 $Re_k(Con_i)$ 的添加而变为真，Con_j 却不会被违背。图 3 中 Con_3 和 Con_{10} 的关系即属于这种情况，因此它们之间的弧是可被删除的非依赖弧。

定理 2 若对于从 $Re_k(Con_i)$ 到 $Pc(Con_j)$ 的所有置换 θ，均存在置换 θ' 满足 $Re_k(Con_i)\theta'\subseteq Pc(Con_j)\backslash Re_k(Con_i)\theta$，其中 $domain(\theta')\subseteq addition(Re_k(Con_i))$，则弧 $Re_k(Con_i)$ 是非依赖弧。

证明：假定例示 $I\not\models Con_i$，通过向 I 中添加 $Re_k(Con_i)$ 的实例进行修复得到的例示为 I'，现通过反证法证明 I' 不可能违背 Con_j。假设 $I'\not\models Con_j$，即 $I'\models Pc(Con_j)$ 且 Con_j 修复集中的任何修复均不被 I' 满足，根据已知条件，对于任意从 $Re_k(Con_i)$ 到 $Pc(Con_j)$ 的置换 θ，均存在置换 θ' 满足 $Re_k(Con_i)\theta\subseteq Pc(Con_j)\backslash Re_k(Con_i)\theta$，可知从 I' 中去除 $Re_k(Con_i)$ 后的例示(实际上就是 I)仍将满足 $Re_k(Con_i)\theta'$，因此可得 $I\models Re_k(Con_i)\theta'$；再考虑 $domain(\theta')\subseteq addition(Re_k(Con_i))$，可知 Con_i 的修复 $Re_k(Con_i)$ 已成立，即 $I\models Con_i$，这与 $I\not\models Con_i$ 相悖，故 Con_i 的修复不可能违背 Con_j。证毕。

图 3 中从 Con_{10} 到 Con_{11} 的弧即属于此种情形，如果 I' 违背 Con_{11}，则将 Con_{11} 左边的两个 Supervises 实例中刚添加进来的(即为了修复 Con_{10} 而添加进来的)那一个去掉之后得到的 I 仍包含一个 $Supervises$ 的实例，该实例给出了 $TeachingGroup(g)$ 的督导，也就是说 g 至少有一名督导的约束得以满足，即 I 满足 Con_{10}，这与 I 不满足 Con_{10} 相悖，因此假设不成立，即 I' 不可能违背 Con_{11}。

定理 3 如果变元 $x\in addition(Re_k(Con_i))$ 的定义域是无穷的，$Pc(Con_j)$ 中存在原子 $P(\bar{t})$ 使得 $assignmentupper$

$(Re_k(Con_i), P(\bar{t}), x) < assignmentlower(Pc(Con_j), P(\bar{t}), x)$，则弧 $Re_k(Con_i)$ 是非依赖弧。其中 $assignmentupper(\varphi, P(\bar{t}), x)/assignmentlower(\varphi, P(\bar{t}), x)$ 表示针对给定文字的集合 φ，为了确保包含变元 x 的原子 $P(\bar{t})$ 的每个文字都为真，需要赋值的闭原子的最大/最小个数。

证明：假定例示 $I \not\models Con_i$，通过向 I 中添加 $Re_k(Con_i)$ 的实例修复 Con_i 后得到的例示为 I'。该修复为变元 x 赋予新值 A（由于 x 的定义域是无穷的，因此 A 总是存在的），现通过反证法证明 $I' \vdash Con_j$。

假设 $I' \not\models Con_j$，即 $I' \vdash Pc(Con_j)$ 且 Con_j 修复集中的任何修复均不被 I' 满足，此时 I' 至少包含 $assignmentlower(Pc(Con_j), P, x)$ 个原子 $P(\bar{t})$ 的实例，其中 x 取值为 A。由于 A 是新值，因此 I 中不包含 x 取值为 A 的 $P(\bar{t})$ 的实例，也就是说 I' 中 x 取值为 A 的 $P(\bar{t})$ 的实例只能是由修复 $Re_k(Con_i)$ 所引入的，即其个数 $= assignmentupper(Re_k(Con_i), P(\bar{t}), x)$。根据 $I' \not\models Con_j$ 可得 $assignmentlower(Pc(Con_j), P(\bar{t}), x) <= assignmentupper(Re_k(Con_i), P(\bar{t}), x)$，这与前提条件 $assignmentupper(Re_k(Con_i), P(\bar{t}), x) < assignmentlower(Pc(Con_j), P(\bar{t}), x)$ 相矛盾，假设不成立，故而 I' 必然满足 Con_j。证毕。

此处对定理 3 中的 $assignmentupper(\varphi, P(\bar{t}), x)$ 和 $assignmentlower(\varphi, P(\bar{t}), x)$ 举例如下：假定 $\varphi = P(x,u) \wedge P(x,v) \wedge P(x,w)$ 且已知 $u \neq v$，则 $assignmentupper(\varphi, P(\bar{t}), x) = 3$（此时 u, v, w 均不相等），而 $assignmentlower(\varphi, P(\bar{t}), x) = 2$（此时 $w = u$ 或 $w = v$））。直观地说，定理 3 描述的是这样的情形：由于向 I 中添加 $Re_k(Con_i)$ 修复 Con_i 时为变元 x 赋予的值不与已有的值重复，所以将永远不会违背 Con_j。图 3 中从 Con_{16} 到 Con_9 的弧即属于此种情形。由于仅当 2 个不同的 *Member* 的实例关联到同一个 *Teacher* 的实例 t 时 Con_9 才可能被违背，而每次通过添加 $\{Member(m, t_1, g), Member(n, t_2, g), Recruits(m, n)\}$ 修复 Con_{16} 时，2 个 *Member* 的实例关联的 *Teacher* 的实例 t_1 和 t_2 并不相同，所以 Con_{16} 的修复永远不会导致 Con_9 被违背。

按照定理 1－定理 3 叙述的 3 个策略，消除图 3 中的非依赖弧，可以得到约简后的约束依赖图，如图 5 所示。

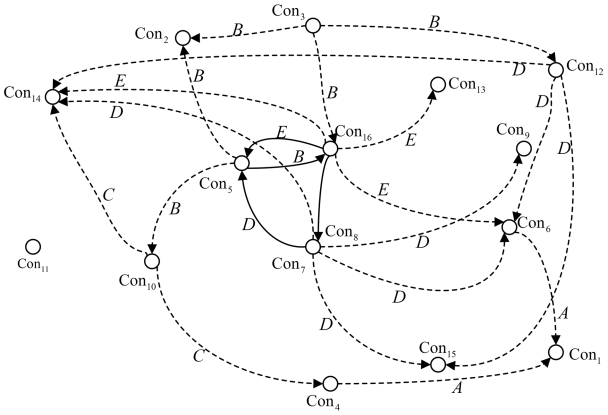


图 5 约简后的约束依赖图（例 1）

Fig. 5 Reduced constraint dependency graph(example 1)

5 有限环路的识别

对于不构成环路的约束，这样的约束一旦被修复之后不会再次被违背或者仅能再次被违背有限次（由于例示中的事实集是有限的），因此不会产生无限的修复序列；相反，对于构成环路的约束，由于修复这样的约束时向例示中添加的事实可能导致该约束再次被违背，因此可能会产生无限次数的修复。根据上述分析可以看出，约束依赖图中存在环路是产生无限修复的必要、非充分条件。

令 $C = (Con_1 Re_1 \cdots Con_n Re_n Con_{n+1} = Con_1)$ 是约束依赖图中构成环路的顶点和弧的交替序列。C 的存在意味着 Con_i 的修复 Re_i 可能违背其他约束，而被违背的约束的修复可能再次违背 Con_i。如果环路 C 的所有运行实例都是顶点和弧的有限交替序列，则称 C 为有限环路。如果一个 $RDFS_{(c)}$ 模式对应的约束依赖图中包含的所有环路都是有限环路，则在该图中的规约检测过程必然会在有限步骤内结束，即面向该 $RDFS_{(c)}$ 模式的验证任务是可判定的。需要特别指出的是，如果一个环路 C 包含非有限子环，那么显然 C 也不是有限环路，因此我们将总是先对子环进行考察，只有当所有的子环都是有限环路时，我们才会对 C 进行考察。基于这个原因，后文所考察的环路中的所有子环（如果存在的话）均已确定为有限环路。本节将研究并提出识别约束依赖图中的有限环路的策略。这些策略由定理 4－定理 6 给出。

定理 4 给定环路 $C = (Con_1 Re_1 \cdots Con_n Re_n Con_{n+1} = Con_1)$，如果 $Pc(Con_i)$ 中存在原子 $Q(\bar{t})$ 使得 $Q(\bar{t}) \notin \bigcup_{j=1}^{n} Re_j$ 并且 $var(Pc(Con_i) \backslash Q(\bar{t})) \subseteq var(Q(\bar{t}))$，其中 $1 \leqslant i, j \leqslant n$，则 C 是有限环路。

证明：令 I 表示模式的初始例示，即执行修复之前的例示，$\theta_1, \cdots, \theta_k$ 表示使得 $Q(\bar{t})\ \theta_i \in I (1 \leqslant i \leqslant k)$ 的 k 个闭置换。用反证法，假设修复 C 中的约束导致了无限的修复序列，则 Con_i 必被违背了无限次。令 I' 是对 Con_i 执行了 k 次修复之后得到的例示，可知 $I' \not\models Con_i$，进而令 θ_{k+1} 是使得 $Pc(Con_i)\ \theta_{k+1} \subseteq I'$ 的闭置换。根据 $Q(\bar{t}) \notin \bigcup_{j=1}^{n} Re_j$ 可知不存在 θ'_{+1} 满足 $Re_i\theta_{k+1} \subseteq I'$，其中 $\theta'_{k+1} = \theta_{k+1} \cup \theta_r$ 而 θ_r 是 Re_i 中的增量变元。根据 $var(Pc(Con_i) \backslash Q(\bar{t})) \subseteq var(Q(\bar{t}))$ 可知，必定存在闭置换 θ_j 使得 $\theta_j = \theta_{k+1}$ 且 $Q(\bar{t})\theta_j \subseteq I'$。由于 Con_i 已经被修复了 k 次，所以 $Pc(Con_i)\ \theta_j \subseteq I'$，进而可得 $Re_i\theta_j \subseteq I'$，其中 $\theta_j' = \theta_j \cup \theta_r$。根据 $\theta_j = \theta_{k+1}$，我们发现存在闭置换 θ'_{k+1} 满足 $Re_i\theta'_{k+1} \subseteq I'$，这与不存在闭置换 θ'_{k+1} 满足 $Re_i\theta'_{k+1} \subseteq I'$ 相矛盾，假设不成立，故而 C 是有限环路。证毕。

直观地讲，定理 4 描述的是这样的情形：约束 Con_i 拥有属于环路中的所有修复以外的潜在违背，因此，由于初始例示中的事实集是有限的，Con_i 至多会被违背有限次。图 6 中的 2 个约束构成的环路即属于这种情形。Con_1 的潜在违背 $Q(x)$ 不属于 Con_2 的修复并且由于 $var(P(x)) \subseteq var(Q(x))$，$Con_1$ 的违背依赖于该原子，因此即便 $P(x) \wedge Q(x)$ 在例示中成立导致了 Con_1 被违背，Con_2 的后续的修复将只导致 Con_1 再次被违背有限次，每次违背由修复过程起始时刻的初始例示中的一个事实 $Q(*)$ 所触发。

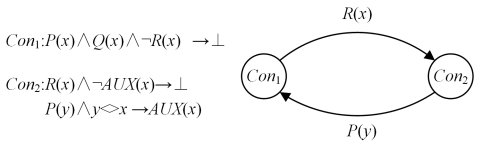


图 6 有限环路的用例

Fig. 6 Example of finite cycle

定理 5 给定环路 $C=(Con_1 Re_1 \cdots Con_n Re_n Con_{n+1}=Con_1)$，对于每一对 $P(x_1,\cdots,x_m)\in Re_i$ 和 $P(y_1,\cdots,y_m)\in Pc(Con_{i+1})$，$1\leqslant i\leqslant n$，如果 x_k 是 Re_i 中的增量变元意味着 $y_k\notin var(Re_{i+1})$，$1\leqslant k\leqslant m$，则 C 是有限环路。

证明：令 I 表示模式的初始例示，k 表示 I 中的不同常量的个数。分 2 种情况讨论：

(1)C 中的任何修复都不包含增量变元，此时任何修复都不会向 I 中添加新的常量，因此 $Pc(Con_i)$ 的不同实例的个数(即约束 Con_i 的不同实例的个数)可以通过 k 个常量的排列组合来求得，而该值必定是有限的，因此 Con_i 被违背的次数必定是有限的。

(2)C 中存在包含增量变元的修复，用反证法，假设 k 个常量导致 C 中的约束被违背无限次并且 Re_i 包含增量变元 x_k。令 I' 是对 C 中除了 Con_i 以外的所有约束执行修复之后得到的例示，那么对于所有 $Con_j\in C, j\neq i$，如果存在置换 θ_j 使得 $Pc(Con_j)\ \theta_j\subseteq I'$，则必存在置换 $\theta_j'=\theta_j\cup\theta_r$ 使得 $Re_j\theta_j'\subseteq I'$，其中 θ_r 是对 Re_j 包含的增量变元进行的置换。令 $P(x_1,\cdots,x_k,\cdots,x_m)$ 表示包含增量变元 x_k 的原子，根据假设 C 中的所有约束被违背无限次，令 $P(x_1,\cdots,x_k,\cdots,x_m)\theta_i=P(y_1,\cdots,y_k,\cdots,y_m)\in Pc(Con_{i+1})$，可知必存在为增量变元 y_k 赋予新常量的置换 θ_{i+1} 使得 $Pc(Con_{i+1})\ \theta_{i+1}\subseteq I'$。再根据前提条件 $x_k\in var(Re_i)$ 而 $y_k\notin var(Re_{i+1})$ 可知，不存在 θ_{r+1} 使得 $Re_{i+1}\theta_{r+1}\subseteq I'$。由于 Con_j 是 C 中除 Con_i 以外的任意约束，自然也包括 Con_{i+1}，所以不存在置换使得 Re_{i+1}(即 Re_j)经过置换后包含于 I'，这与前述 $Re_j\theta_j'\subseteq I'$ 相矛盾，因此 C 中的约束被违背的次数必定是有限的。证毕。

定理 5 指出，对于所有为了修复 Con_i 所引入的增量变元 x_k，其置换后生成的 y_k 均不出现在 Con_{i+1} 的修复集中，满足这个条件可以避免在修复过程中无限地引入新的元素，因此该环路是有限的。将其应用于图 5，我们发现 $Con_5-Con_{16}-Con_5$ 是有限环路，因为 Con_5 的修复并不引入增量变元，尽管 Con_{16} 的修复引入了增量变元 m、n、t_1 和 t_2，但它们都不出现在 Con_5 的修复集中，从而避免了新实例的传播。

定理 6 给定环路 $C=(Con_1 Re_1 \cdots Con_n Re_n Con_{n+1}=Con_1)$，令 $Facts(Con_i,Con_j)=$

$$\begin{cases}\bigcup_{k=1}^{t}(Pc(Con_i)\cup Re_i)\theta_k, & \text{如果 } i=j\\ \bigcup_{k=1}^{t}(Facts(Con_i,Con_{j-1})\cup Re_j)\theta_k, & \text{如果 } i\neq j\end{cases}$$

，其中 θ_k 是为每个变元赋值的闭置换，t 是不同的 θ_k 的个数，如果对于任意的 i，均存在 j 使得 $Facts(Con_i,Con_j)=Facts(Con_i,Con_{j+1})$，则 C 是有限环路。

证明：(用反证法)假设对于任意的 i，均存在 j 使得 $Facts(Con_i,Con_j)=Facts(Con_i,Con_{j+1})$，但 C 被违背无限次。首先令 $i=1$，分 2 种情况进行讨论：

(1)不存在 θ 使得 $Pc(Con_{j+1}\cup Re_{j+1})\theta\subseteq Facts(Con_1,Con_j)$，也就是说，此时对于每个 $P(\bar{t})\in Pc(Con_{j+1})$，$P(\bar{t})$ 不存在于从 Con_1 到 Con_j 的任何一个修复中，即 $P(\bar{t})\notin(Re_1\cup Re_2\cup\cdots\cup Re_j)$。此时 Con_{j+1} 被违背无限次的唯一可能是存在 $m,j+1\leqslant m\leqslant n$ 且 $P(\bar{t})\in Re_m$，也就是说存在从 Con_m 到 Con_{j+1} 的弧 Re_m，该弧使得 $C'=(Con_{j+1}Re_{j+1}\cdots Con_m Re_m Con_{m+1}=Con_{j+1})$ 构成了 C 的一个子环。如前所述，只有当子环都是有限环路的情况下我们才会考察 C 是否是有限的，因此 C' 必定是有限环路，即 Re_m 并不会导致无限个 $P(\bar{t})$ 实例的引入，那么引入无限个 $P(\bar{t})$ 实例的修复只能是 Re_j，即 $P(\bar{t})\in Re_j$，这与 $P(\bar{t})\notin(Re_1\cup Re_2\cup\cdots\cup Re_j)$ 相矛盾，因此 C 不可能被违背无限次。

(2)存在 θ 使得 $Pc(Con_{j+1}\cup Re_{j+1})\theta\subseteq Facts(Con_1,Con_j)$，根据 Con_{j+1} 被违背无限次可知，此时必定存在某个 $Con_k\neq Con_{j+1}$ $(1\leqslant k\leqslant j)$ 使得 $Pc(Con_k)\theta\subseteq Facts(Con_1,Con_j)$ 并且不存在 θ' 使得 $Pc(Con_k\cup Re_k)\ \theta'\subseteq Facts(Con_1,Con_j)$，也就是说存在从 Con_{k+1} 到 Con_1 的弧 Re_{k+1}，而该弧使得 $C'=(Con_1Re_1\cdots Con_{k+1}Re_{k+1}\ Con_{k+2}=Con_1)$ 构成了 C 的一个子环。由于 C' 是有限环路，因此引入无限个实例的修复只能存在于 C 以外，即由介于 C 中的 Con_{k+2} 和 Con_n 之间的约束所引入，令其为 Con_m，$k+2\leqslant m\leqslant n$。进而可知 $Facts(Con_1,Con_j)\subset Facts(Con_1,Con_m)$，再根据 $Facts(Con_1,Con_m)\subseteq Facts(Con_1,Con_{j+1})$ 可得 $Facts(Con_1,Con_j)\subset Facts(Con_1,Con_{j+1})$，这与前提 $Facts(Con_1,Con_j)=Facts(Con_1,Con_{j+1})$ 相悖。

综上，当 $i=1$ 时，C 必定被违背有限次。$i=2,3,\cdots,j$ 时的证明方式与之类似，此处不再赘述。最终可得，对于任意的 i，如果均存在 j 使得 $Facts(Con_i,Con_j)=Facts(Con_i,Con_{j+1})$，则 C 必定是有限环路。证毕。

定理 6 指出，采用 C 中的约束之一作为起点，向 $Facts$ 中添加它的潜在违背和相应修复的实例。这些实例为该约束的潜在违背和相应修复中的每个变元逐次分配所有的新常量。这一过程将导致最大数目的新实例被建立，从而导致最大数目的违背发生。接着，修复随后被违背的约束，方法是为相应修复中的每个变元逐次指派所有可能的新常量，并将这些修复添加进 $Facts$ 中。如果无论采用 C 中的哪个约束作为起点，当该约束再次被修复之前，$Facts$ 中的元素数目均停止增长，则意味着 C 中约束的违背必然是有限的。将之应用于图 5，我们发现 $Con_{16}-Con_7-Con_5-Con_{16}$ 和 $Con_{16}-Con_8-Con_5-Con_{16}$ 均是符合此种情形的有限环路。

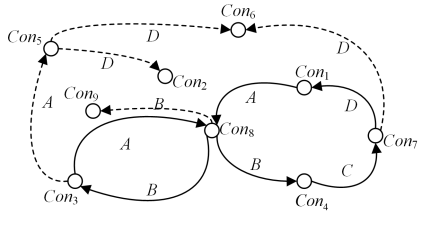
将上述理论应用于例 1，对于图 5 中的所有 3 个环路(Con_7 和 Con_8 各参与 1 个环路)，由于每个环路至少满足定理 4－定理 6 其中之一，因此可以得出结论：在该图对应的 $RDFS_{(c)}$ 模式之上的验证任务是可判定的。对于例 2 来说，利用表 1 中的方法对 $RDFS_{(c)}$ 模式进行形式化，可以得到表 3 所列的结果。

表 3 图 2RDFS$_{(c)}$模式对应的一阶约束

Table 3 First-order constraints corresponding to RDFS$_{(c)}$ schema in Fig. 2

图形约束	对象属性的完整性
	con1: IsManagedBy(d,s) ∧ ¬Division(d) → ⊥
	con2: IsManagedBy(d,s) ∧ ¬Staff(s) → ⊥
	con3: HasStaff(d,s) ∧¬Division(d) → ⊥
	con4: HasStaff(d,s) ∧ ¬Staff(s) → ⊥
非图形约束	约束 1
	con5: Division(d) ∧ ¬oneManager(d) → ⊥ IsManagedBy(d,s) → oneManager(d)
	con6: IsManagedBy(d,s1) ∧IsManagedBy(d,s2) ∧s1〈〉s2→ ⊥
	约束 2
	con7: Staff(s) ∧ ¬ oneManagedDivision(s)→ ⊥ IsManagedBy(d,s) → oneManagedDivision(s)
	约束 3
	con8: Division(d) ∧ ¬lowerStaff(d) → ⊥ HasStaff(d,s1) ∧ HasStaff(d,s2) ∧ s1〈〉s2 → lowerStaff(d)
	约束 4
	con9: HasStaff(d1,s) ∧ HasStaff(d2,s) ∧ d1〈〉d2→ ⊥

基于表 3 建立约束依赖图并利用定理 1－定理 3 叙述的 3 个策略消除非依赖弧之后，得到的结果如图 7 所示。



A:{Division(d)} B:{HasStaff(d,s)} C:{Staff(s)} D:{IsManagedBy(d,s)}

图 7 约简后的约束依赖图(例 2)

Fig. 7 Reduced constraint dependency graph (example 2)

图 7 共包括 2 个环路，根据定理 5 可知 $Con_3-Con_8-Con_3$是有限环路，原因是 Con_3的修复并不引入增量变元，而 Con_8的修复中的增量变元并没有被传播到 Con_3的修复集中。根据定理 4 和定理 5，另一个环路 $Con_1-Con_8-Con_4-Con_7-Con_1$不是有限环路。下面对其应用定理 6。先选择 Con_1作为起点来模拟 *Facts* 的变化：$Facts(Con_1,Con_1)=\{IsManagedBy(1,2),Division(1)\}$；$Facts(Con_1,Con_8)=\{IsManagedBy(1,2),Division(1),HasStaff(1,3),HasStaff(1,4)\}$；$Facts(Con_1,Con_4)=\{IsManagedBy(1,2),Division(1),HasStaff(1,3),HasStaff(1,4),Staff(3),Staff(4)\}$；$Facts(Con_1,Con_7)=\{IsManagedBy(1,2),Division(1),HasStaff(1,3),HasStaff(1,4),Staff(3),Staff(4),IsManagedBy(5,3),IsManagedBy(6,4)\}$；$Facts(Con_1,Con_1)=\{IsManagedBy(1,2),Division(1),HasStaff(1,3),HasStaff(1,4),Staff(3),Staff(4),IsManagedBy(5,3),IsManagedBy(6,4),Division(5),Division(6)\}$。可见，*Facts* 中的元素数目在起点再次被修复前并未停止增长。选择其他约束作为起点，可以得到同样的结果。因此根据定理 6，$Con_1-Con_8-Con_4-Con_7-Con_1$不是有限环路，进而可知在例 2 之上的模式验证任务不是可判定的。

6 基于约束依赖性解析的 RDFS$_{(c)}$ 模式验证算法

下面基于约束依赖关系之上的推理来检测与 STC 对应的一阶表达式的可满足性，进而对模式进行验证。验证算法的基本思路如下：该算法的目的是构建一个使得 STC 成立的例示 I，使得模式中的所有约束在 I 中都得到满足。该过程分为以下两个阶段：规约满足阶段和约束满足阶段。

在规约满足阶段，确定满足特定 *STC* 所需的 I 中的事实，而不考虑这些事实是否违背在模式的逻辑形式化中所定义的、用来表示图形约束和非图形约束的一阶约束。如 3.2 节所述，每个 STC 都被指定为文字和条件表达式的合取，其中的肯定文字表示满足该规约所必需的事实，而否定文字则对应于在构建的 I 中所禁止的事实。条件表达式说明了对肯定文字和否定文字中的变元可能采用的值的要求。在将具体值指派给要实例化的文字中的变元时，由于 RDFS$_{(c)}$ 模式中的初始事实是有限的，因此构建的例示 I 中的元素数目必然是有限的。

在约束满足阶段，必须确保所获得的例示不违背模式中的任何约束。依赖图提供了应该考虑约束的顺序。没有输入弧或其输入弧已被处理过的顶点被优先选择，以便在所有可能违反该约束的约束都已经被修复之后，才会考虑该约束。如果所有顶点都有待处理的输入弧，则优先选择输入弧数目少的顶点。如果约束 *con* 的潜在违背在 I 中成立，则必须修复 *con*。然后将它的修复集加入正在构建的例示 I 中。需要注意的是，*con* 可能会被其潜在违背的几个不同指派所违背，此时每个指派都将导致不同的修复实例被添加到 I 中。当修复失败时，即生成的例示 I 违背某个约束，而该约束无法被进一步修复时，将使用下一个置换对该修复进行指派。当该修复的所有实例都已被尝试并且失败时，如果该约束还有其他修复，将依次选择其他修复。最后，当约束 *con* 的所有修复都失败，或者当 *con* 的修复集为空时，必须丢弃正在构建的例示 I，因为不可能使其满足 *con*。随后向前回溯，考虑 *con* 的前一个约束的修复的下一种指派(采用下一个置换)和下一种修复，如同刚才所解释的。

重复上述过程，如果最终无法得到满足所有约束的例示 I，则说明与 STC 对应的一阶公式是不可满足的，即在所定义的 RDFS$_{(c)}$ 模式之上，该规约是不可能被确保的；相反，当被构造的例示 I 不违背任何约束时过程终止，那么获得的 I 证明该模式满足该规约。算法的详细步骤如算法 1 所示。

算法 1 检测 STC 的可满足性

输入：约束依赖图 G，$STC=P_1(\bar{x}_1)\wedge\cdots\wedge P_n(\bar{x}_n)\wedge\neg Q_1(\bar{y}_1)\wedge\cdots\wedge\neg Q_m(\bar{y}_m)\wedge b_1\wedge\cdots\wedge b_s$，其中 P、Q 均为谓词名且 Q 非由演绎规则所定义，$b_k(1\leqslant k\leqslant s)$为 STC 中的条件表达式

输出：STC 的可满足性结果，如果可满足，附加输出满足 STC 和所有约束的一个例示 I.

1. 初始化例示 I=∅，为满足 STC 所需要的事实集 FN=∅，为满足 STC 所禁止的事实集 FP=∅；
2. 对于每个 $b_k(1\leqslant k\leqslant s)$，执行：{
3. 计算使得 $b_k\theta$ 为真的闭置换 θ 的个数，令其为 t，对于每个 $\theta_i(0\leqslant$

i≤t)执行：{

4. $FN_i=\{P_1(\bar{x}_1)\theta_i,\cdots,P_n(\bar{x}_n)\theta_i\}$；

5. $FP_i=\{Q_1(\bar{y}_1)\theta_i,\cdots,Q_m(\bar{y}_m)\theta_i\}$；

6. }；

7.}；

8.将 G 中的所有节点按输入弧个数从小到大的顺序进行排序，并建立队列 L；

9.依次取出 L 中的每个元素，将其标记为 con，执行下述操作：

10)对于每个 FN_i(0≤i≤t)，若 FN_i 违背 con，则：

11. {取出 con 的第一个修复，将其标记为 Re(con)；

12. 若 Re(con)=∅，则：

13. {从 L 中取出 con 的前一个元素，若为空，则跳转到步骤(23)；

14. 若不为空，则将其标记为 con，跳转到步骤(10)；}

15. 否则：

16. $\{FN_{i+1}=FN_{i+1}\quad Re(con)\theta_i$；

17. 若 $\exists Q_j(\bar{y}_j)\theta_{i+1}\in FP_{i+1}$(1≤j≤m)使得 $FN_{i+1}\nvdash Q_j(\bar{y}_j)\theta_{i+1}$ 或者 FN_{i+1} 违背 con，则：

18. $\{FN_{i+1}=FN_{i+1}\quad Re(con)\theta_{i+1}$；

19. 跳转到步骤 10；}

20. 否则：

21. {取出 con 的下一个修复，若为空，则跳转到步骤(23)；

22. 若不为空，则将其标记为 Re(con)，跳转到步骤(12)；}}}

23. 计算 $I=\bigcup_{i=0}^{t}FN_i$；

24.依次取出 L 中的每个元素，将其标记为 con，执行下述操作：

25.若 I 违背 con，且 Re(con)=∅ 或不存在 $Re_j(con)\in Re(con)$ 使得 $I\vdash Re_j(con)\theta_i$，算法结束，输出“STC 是不可满足的”

26.算法结束，输出“STC 是可满足的”并输出 I。

7 实验及相关工作比较

为了证明本文方法的有效性，我们实现了一个原型系统 RdfsCV。系统以 RDFS 格式文件以及包含非图形约束的文本文件作为输入。转换组件负责将 $RDFS_{(c)}$ 模式转换为一阶约束。可判定性组件负责推断在输入模式之上的验证任务的可判定性，最后由推理组件对模式进行验证。一旦模式被转换为一阶约束，可判定性组件就能够构建依赖图并对其进行解析。如果确定图中找到的所有环路都是有限的，那么模式验证任务会自动执行，并保证它会终止。RdfsCV 会报告哪些类和属性可满足或不可满足，哪些约束冗余或非冗余。

下面和相关工作进行定性比较。从 4 个方面进行对比：支持约束建模语言的种类，支持的约束种类，可验证的规约，是否支持可判定性的推断。详细结果列于表 4 中。可以看出，Ontolo-CI 支持的约束和规约的种类最少，适用范围较窄。DLV2，FHIR 和 FHIR 均不支持可判定性的推断，从而无法确保模式验证任务是可终止的。相比之下，本文的方法采用一阶谓词逻辑作为约束建模语言，其强大的表达能力赋予了方法最大的适用性，特别是如果建立了 ShEx 语言和 SHACL 语言到一阶谓词逻辑的映射，可以将上述 3 个工具的验证任务转移到我们的工具中，并且支持可判定性的推断。

表 4 相关工作的定性比较

Table 4 Qualitative comparison of related works

约束建模语言		支持的约束	支持的规约	推断可判定性
DLV2	SHACL	类层次 属性的完整性 类属对象属性的实例唯一性 基数约束	类的可满足性 属性的可满足性	不支持
FHIR	ShEx	类层次 属性的完整性 类属对象属性的实例唯一性 基数约束	类的可满足性 属性的可满足性 约束的非冗余性	不支持
Ontolo-CI	ShEx	类层次 属性的完整性 类属对象属性的实例唯一性	类的可满足性 属性的可满足性	不支持
RdfsCV	一阶谓词逻辑	类层次 属性的完整性 类属对象属性的实例唯一性 基数约束	类的可满足性 属性的可满足性 约束的非冗余性	支持

为了对比 RdfsCV 和相关工具的时间性能，执行了超过 40 次的实验。实验选取 LUBM-50 和 LUBM-100 作为数据集。对于每个数据集，分别随机选择 4 个元素并执行规约的验证。由于本文的方法是针对转换得到的一阶表达式提出了专门的可满足性检测算法，从而完成规约的验证，而一阶逻辑的通用定理证明器(如 iProver 和 Leanprover)也可以执行可满足性检测任务，因此将二者也纳入比较的范围。所有实验是在 Intel 酷睿 i5-1155G7、16 GB 内存和 Windows 10 操作系统的环境下执行的。表 5 和表 6 列出了各工具的执行时间，均以秒计。在 RdfsCV 执行时间的后面，还给出了所构建的例示中包含的实例数目。

表 5 LUBM-50 实验结果

Table 5 Experimental results of LUBM-50

	Graduate Course	Emeritus Scholar	takes Course	C-11-3 for Responsible Party
RdfsCV	0.82(16)	1.12(23)	1.76(48)	9.34(98)
DLV2	0.56	0.89	1.62	—
FHIR	3.24	2.94	3.76	12.02
Ontolo-CI	2.76	2.62	3.54	—
iProver	6.12	5.27	8.39	18.80
Leanprover	6.92	5.23	7.74	16.51

表 6 LUBM-100 实验结果

Table 6 Experimental results of LUBM-100

	Staff Member	Social Service	hasWeekly Breakup	C-5-16 for BehavioralFeature
RdfsCV	0.94(18)	1.37(27)	1.74(44)	12.82(112)
DLV2	0.47	0.64	1.61	—
FHIR	3.30	3.12	4.54	11.91
Ontolo-CI	2.01	2.97	3.12	—
iProver	8.22	7.10	9.37	17.77
Leanprover	8.19	7.23	9.98	20.52

对于 LUBM-50，选择 3 个元素，即类 GraduateCourse、类 EmeritusScholar、对象属性 takesCourse 执行可满足性检测，并对约束 C-11-3 for ResponsibleParty 执行非冗余性检测。对于前 3 个任务，RdfsCV 的平均执行时间是 1.23 s，所构建的例示中包含的模式元素的实例数目平均为 29 个。而 DLV2，FHIR 和 Ontolo-CI 的平均执行时间分别为 1.02 s，3.31 s 和 2.97 s。如果把约束非冗余性检测也考虑进来，则 RdfsCV 和 FHIR 的平均执行时间分别为 3.26 s 和 5.49 s。总体而言，本文的方法取得了略逊于 DLV2 但显著优于 FHIR 和 Ontolo-CI 的时间性能。相比之下，iProver 和 Leanprover 耗时最长，4 个任务的平均执行时间达到 9.65 s 和 9.10 s。究其原因，为了确保符合条件的例示一定会被找到(如果存在)，RdfsCV 会尝试所有可能的组合来构建符合条件的例示，但只要找到一个即成功退出，因此并不会降低检测效率。更重要的是，约束依赖性解析确保了在某个约束被处理的时候，所有可能重复违背该约束的约束都已被修复，因而最大程度地减少了回溯次数，从而显著提高了检测效率。本文方法在 LUBM-100 上取得了类似的实验结果，此处不再赘述。

结束语 为了推导 $RDFS_{(c)}$ 模式验证问题的可判定性并高效地对模式进行验证，本文提出了一种基于约束依赖性解析的方法。将 $RDFS_{(c)}$ 模式及待检测的规约统一于一阶谓词逻辑框架下，分析了约束之间的修复-违背关系并识别约束依赖图中的有限环路，在此基础上提出了一种模式验证算法。该方法使得我们可以在模式验证之前预知哪些规约检测任务是可终止的。实验结果表明，本方法在保证了强适用性的前提下，具备相对较好的时间性能。未来的工作包括 2 个方面。(1)增强本方法的可解释性，以便追溯模式验证中发现的错误的原因，从而为修复错误提供支持。由于约束依赖性解析能够直观地反映出待检测规约不满足哪些约束，因此本方法在这方面有天然的优势。(2)研究 ShEx 语言和 SHACL 语言到一阶谓词逻辑的映射方法，从而将 DLV2 和 FHIR 的验证任务纳入本方法的框架并支持可判定性的推断，这也是一个有趣的研究思路。

参 考 文 献

[1] TOMASZUK D, HAUDEBOURG T. RDF Schema 1.2. W3C Recommendation [EB/OL]. www.w3.org/TR/2023/WD-rdf12-schema-20230615.

[2] JOSE E L, HERMINIO G, DANIEL F, et al. Challenges in RDF Validation[C]// Current Trends in Semantic Web Technologies: Theory and Practice. Berlin: Springer-Verlag, 2019: 121-151.

[3] MEESTER B D, HEYVAERT P, ARNDT D, et al. RDF graph validation using rule-based reasoning[J]. Semantic Web, 2021, 12(1): 117-142.

[4] WEYNS M, BONTE P, STEENWINCKEL B, et al. Conditional Constraints for Knowledge Graph Embeddings[C]// Proceedings of the 17th Extended Semantic Web Conference (ESWC 2020). Berlin: Springer-Verlag, 2020: 1-10.

[5] LIEBER S, DEMEESTER B, HEYVAERT P. Visual notations for viewing RDF constraints with UnSHACLed [J]. Semantic Web, 2022, 13(5): 757-792.

[6] LE N L, ABEL M H, GOUSPILLOU P. A Constraint-based Recommender System via RDF Knowledge Graphs[C]// Proceedings of the 26th International Conference on Computer Supported Cooperative Work in Design (CSCWD2023). New York: IEEE Press, 2023: 849-854.

[7] NANDANA M. A framework for linked data quality based on data profiling and rdf shape induction [D]. Madrid: Universidad Politécnica de Madrid, 2020.

[8] CHMUROVIC A, SIMKUS M. Well-founded Semantics for Recursive SHACL[C]// Proceedings of the 4th International Workshop on the Resurgence of Datalog in Academia and Industry. Berlin: Springer-Verlag, 2022: 2-13.

[9] GARCIA-GONZALEZ H, LABRA-GAYO J E. XML Schema2ShEx: Converting XML validation to RDF validation [J]. Semantic Web, 2020, 11(2): 235-253.

[10] FERNANDEZ-ALVAREZD, LABRA-GAYOJE, GAYO-AVELLO D. Automatic extraction of shapes using sheXer [J]. Knowledge-Based Systems, 2022, 238: 107975.

[11] ROBALDO L, PACENZA F, ZANGARI J, et al. Efficient compliance checking of RDF data[J]. Journal of Logic and Computation, 2023, 34(6): 98-121.

[12] PRUDHOMMEAUX E, COLLINS J, BOOTH D, et al. Development of a FHIR RDF data transformation and validation framework and its evaluation[J]. Journal of Biomedical Informatics, 2021, 117: 103755.

[13] PUBLIO G C, GAYO J E L, COLUNGA G F. Ontolo-CI: Continuous Data Validation With ShEx[C]// Proceedings of the 18th International Conference on Semantic Systems. Germany: CEUR-WS.org, 2022: 32-37.

[14] LENZERINI M, NOBILI P. On the satisfiability of dependency constraints in entity-relationship schemata[C]// Proceedings of the 13th International Conference on Very Large Databases (VLDB 1987). Brighton: Morgan Kaufmann, 1987: 147-154.

[15] FITTING M. First-order logic and automated theorem proving [M]. Berlin: Springer Science & Business Media, 2012.

ZHAO Xiaofei, born in 1978, Ph.D, associate professor. His main research interests include knowledge engineering and distributed intelligence.

(责任编辑：柯颖)