论文分类号： TP39　　单位代码： 10183
研究生学号： 2004532157

吉　林　大　学

硕　士　学　位　论　文

基于OWL描述的本体推理研究

Research on Ontology Reasoning Based on OWL

作者姓名：龚　资
专　业：计算机应用技术
导师姓名：欧阳继红
及 职 称：教　授

学 位 类 别：工学硕士
论文起止年月：2005 年 10 月至 2007 年 4 月

吉林大学硕士学位论文原创性声明

本人郑重声明：所呈交的硕士学位论文，是本人在指导教师的指导下，独立进行研究工作所取得的成果。除文中已经注明引用的内容外，本论文不包含任何其他个人或集体已经发表或撰写过的作品成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式标明。本人完全意识到本声明的法律结果由本人承担。

学位论文作者签名：

日期：2007 年 4 月 26 日

作者姓名	龚 资	论文分类号	TP39
		研究生学号	2004532157
学位类别	工学硕士	授予学位单位	吉林大学
专业名称	计算机应用技术	培养单位 (院、所、中心)	计算机科学 与技术学院
研究方向	本体推理	学习时间	2004 年 9 月 至 2007 年 6 月
论文中文题目	基于 OWL 描述的本体推理研究		
论文英文题目	Research on Ontology Reasoning based on OWL		
关键词(3-8 个)	本体，RDF，JENA，本体描述语言，OWL 语言，本体推理		

导师情况	姓 名	欧阳继红	职 称	教 授
	学历学位	博士	工作单位	吉林大学

论文提交日期	2007 年 4 月 26 日	答辩日期	2007 年 6 月 日
是否基金资助项目	是	基金类别 及编号	国家自然科学基金重大项目(60496321)， 国家自然科学基金项目(60373098，60573073)

如已经出版，请填写以下内容			
出版地(城市名、省名)		出版者(机构)名称	
出版日期		出版者地址(包括邮编)	

提　要

近年来，关于本体的研究取得了很大进展，本体的应用也越来越广泛。人们可以从本体中获得知识，而在本体中一些重要的知识往往都是隐性给出的，因此，推导出隐性知识就成了现今对本体研究的关键。随着本体的发展，本体描述语言也趋于多样化。自从 2002 年 OWL 语言成为 W3C 推荐的本体描述语言以来，对 OWL 语言的研究越来越多，而从 OWL 语言描述的本体中推导出隐性知识已成为现今本体研究的主要内容之一。

本文通过对 RDF 蕴涵规则的研究和对 OWL 语言算子的分析，给出了推理规则集和 ORBO 算法，实现了对给定本体的推理。具体工作包括：(1)对本体研究进行了综述，对 OWL 语言和编译工具 JENA 进行了详细的介绍；(2)介绍了 RDF 的属性关系和蕴涵规则；(3)研究了 OWL 算子所蕴涵的语义，并给出了这些算子的推理规则；(4) 为了更好地实现对本体的推理，分析了 Li 所提出的 PD*算法的不足，提出了 ORBO 算法，并给出了 ORBO 算法的 ADL 描述；(5)设计并实现了基于 OWL 语言的推理演示系统。

本文给出的 OWL 语言的推理规则是建立在本体的 TBOX 上的，对 OWL 语言来说更具有一般性；ORBO 算法采用判断推理规则前提的是否满足的方法，降低了原算法的时间复杂度。本文的工作对于从本体中获得隐性知识及以后的查询工作研究都具有一定的意义。

关键词：本体，RDF，JENA，本体描述语言，OWL 语言，本体推理

目　录

第一章　绪　　论

1.1　研究背景

本体(Ontology)最早是一个哲学上的概念，从哲学的范畴来说，Ontology 是客观存在的一个系统的解释或说明，关心的是客观现实的抽象本质。近年来本体受到信息科学领域的广泛关注，成为计算机科学中的一种重要方法。尤其最近本体论在 Web 上的应用导致了语义 Web [1,2] 的诞生，在 W3C 的主导下有望解决 Web 信息共享时的语义问题，从而实现世界范围内的知识共享和智能信息集成。

W3C 为了更好适应语义网需要，推出了 OWL(Web Ontology Language)语言[3,4]。OWL 语言目前已成为语义网发展和本体建立的重要工具之一， 它以描述逻辑[5]和框架逻辑为形式基础， 以描述逻辑为语义推理基础。其中的 OWL DL 既提供了丰富的表达能力， 也可以使描述逻辑语义推理的判定性有一定的保证。

为了实现面向机器理解的万维网，Tim Berners-Lee 于 2001 年正式提出了语义 WEB[1](Semantic Web)。语义万维网是当前万维网的扩展，逐渐将 Internet 变成一个巨大的全球化的知识库，这个知识库能够满足人们浏览信息的需要，更重要的是通过标准的语义规范使计算机自动读取和处理信息。

1.1.1　研究意义及目的

推理的一个基本内容就是由给定的知识获得隐性的知识，在本体中的推理从根本上说就是把隐性在显式定义和声明中的知识通过一种处理机制提取出来。但是，盲目的获得隐性的知识对建立和使用本体帮助不大，所以要先分析清楚 Web 本体推理的应用需求，才能有效的组织推理机制。本体的推理有多方面的应用：对于本体的建立者，推理的主要作用是检测冲突，优化表达和本体融合；对于本体的使用者，推理的作用主要是获得本体中的知识和运用本体中的知识解决问题。

OWL (Web 本体语言，Web Ontology Language)是 W3C 推荐的本体

描述语言的标准。它作为一种本体描述语言，本身不具备任何推理和计算能力，它需要一个推理机，只有推理机对它所描述的内容进行推理，它的描述才有意义。

随着 OWL 语言的提出并成为 W3C 的推荐的本体语言标准，用 OWL 语言描述的本体已经越来越多。RDF 描述是 OWL 语言是基础，所以对 RDF 的推理机制对 OWL 语言依然有效，而 OWL 语言本身还有着自己的算子。对 OWL 语言本身的算子的研究可以是我们更好的了解这门语言，并从其中找到更有效的推理规则和一些隐性的推理规则。

RDF 是 W3C 在 XML 的基础上开发的一种标准，用于表示任何的资源信息。RDF 提出了一个简单的模型用来表示任意类型的数据。RDF 的数据模型实质上是一种二元关系的表达，由于任何复杂的关系都可以分解为多个简单的二元关系，因此 RDF 的数据模型可以作为其他任何复杂关系模型的基础模型[5]。

OWL 是从 RDF 上发展起来的本体描述语言，它可以更好的对本体进行描述，并且可以通过它们对本体已经描述的知识得到了解，但是它们却不能对它们描述的内容进行推理，而一些我们想要得到的知识却隐性在其中。这样，我们就必须对本体进行相应的推理，推导出本体中存在，但是被隐性的知识，以满足我们对隐性知识的需求。

推理规则是推理的基础，没有推理规则我们就无法对现有知识进行推理。本文对 RDF 和 OWL 语言的进行研究，对 RDF 语言进行研究，对 OWL 语言的构造算子进行分析，提出了基于 OWL 语言构造算子的新的推理规则。可以更好的推导出本体中隐性的知识。

所以，通过对 RDF 语言的研究，我们可以更好对 OWL 语言进行掌握，并通过 OWL 语言本身的算子，找到 OWL 语言中蕴涵的隐性推理规则，达到我们对本体进行更好的推理的目的。

1.1.2 研究现状

随着语义 WEB 的快速兴起和发展，本体也越来越得到人们的重视，对本体的描述语言也得到了发展。由自上个世纪 90 年代以来的一些基于 AI 的本体实现语言，如 KIF，Ontolingua， CycL，Loom，OCML，

FLogic。发展成现在一系列可以基于 WEB 的本体语言，也叫做本体描述语言，如 SHOE，XOL，RDF，RDF-S，OIL，DAML，DAML+OIL，OWL。特别的 OWL 语言，已经成为 W3C 推荐的本体描述语言。

本体(Ontology)本来是一个哲学上的概念，近二十年来，本体概念广泛应用到计算机领域，用于人工智能研究中的知识表示、共享以及重用。本体是对某一领域的概念及概念之间关系的显式说明。将本体技术应用于知识系统能够为人与计算机系统之间的通讯提供语法或者语义上的标准，并有助于提高系统可重用性，可靠性及知识获取能力[3]。就知识表示而言，本体语言作为描述逻辑语言，兼顾表示能力与推理能力[4]。本体论知识表示方法是目前最得广大知识表示研究者信任的方法，也成为计算机科学中的一种重要方法，在搜索引擎、知识处理平台、异构系统集成、电子商务、自然语言理解、知识工程、信息抽取与检索、多 agent 系统、物理系统的定性建模、数据库设计、地理信息科学和数字图书馆等领域有重要应用，并得到广泛认同。

知识表示——如何最佳地捕捉智能行为的关键特征以供在计算机上使用，或者说以供与人类进行交流——一直是人工智能(Artificial Intelligence，AI)的一个永恒的主题。知识表示方法通常致力于提供对于领域的全面地表示以支持智能应用，也即提供一种表示推理方法使得能够从显性知识推理出隐性知识[6]。

所谓隐性知识是指那些没有一被文字直接表述出来，但隐含在本体内容中或者由其他相关网页提供的一些重要的信息。往往，我们需要的知识都隐含在隐性知识中。如何更好的得到隐性知识也是本体的主要发展方向。

因此，针对知识表示方法的表示语言和推理算法等也就不断涌现：

2002 年，McDermott 等人[7,8,9,10]提出的对于 AI 中的时间的推理。

2002 年，Kowalsky 等[11]提出的对简单本体的推理分析。

2003 年，Ruud wander Pol[12]提出了一种能够根据事物特征描述概念的 Dipe-R 语言。

2004 年，Yuzhong Qu[13]在论文中提出了一种针对于 Web 本体语言的基于一阶逻辑的推理算法。

2004 年，Riccardo Rosati[14]在论文中对本体论知识表示方法的推理方法进行了探讨。

2004 年，Ian Horrocks 等人[15]提出了 FACT++的 role absorption 算法。

2005 年，林仙等[15]根据上下文的局部性原理和一致性原理提出了基于上下文的表示和推理方法。

2006 年，Seiji Koide 和 Hideaki Takeda[16]，在 ASWC（亚洲语义网会议）上对 OWL full 语言的正向推理进行了详细的阐述。

2006 年，Li 等人[17]提出了基于 RDF 和 PD*语义的推理算法。

目前，对本体的研究已经成为人们研究的重点，对本体推理的研究也越来越成熟，更多的研究趋向于领域本体的推理，而对公理集的研究还为数很少。Ian Dickinson[12]等对公理集研究取得了很大进展，但对公理集推理规则含盖不够全面，对其中的一些规则也可以再做改进。对本体推理采用的算法也多种多样，但有些只针对某领域本体开发，而有些则在某些方面如时间复杂度等加以改进。如 Li 提出的 PD*算法，采用的迭带递归过程，时间复杂度很高。本文设计了 ORBO 算法，对 PD*算法加以改进，降低了时间复杂度。

1.2 本文工作

本文分析并研究了 RDF 语言的蕴涵规则、OWL 语言的表示算子，实现了对本体的推理工作。主要工作包括：分析研究了本体的组成及关系、本体描述 RDF 的蕴涵规则；详细介绍了 W3C 推荐的本题描述语言 OWL 的构造算子；在此基础上针对 OWL 的构造算子和描述语言间的关系，提出了一些推理规则，根据这些推理规则构造规则集；在规则集的基础上，提出了 ORBO 算法，并给出了它的 ADL 描述描述；实现对已给定 OWL 描述的本体进行推理。

全文分为五章，具体安排如下：

第一章 主要介绍本文的研究背景，简单介绍了本体推理研究的意义及目的、研究现状。

第二章 具体介绍了和本文密切相关的理论基础知识，给出了 OWL 语言的一些类和属性，和编程语言 JENA 的语法格式。

第三章 围绕 RDF 语义和描述语言 OWL 的构造算子，根据 OWL 语言中的约束类和属性，提出了 OWL 的推理规则。

第四章 分析了 PD*算法的不足之处，给出了 ORBO 算法。

第五章 设计并实现了基于 OWL 描述的本体推理演示系统。

第六章 对全文工作的总结，并对下一步的研究工作进行了展望。

第二章 本体及本体相关语言介绍

2.1 本体的介绍

2.1.1 Ontology 的定义

Ontology 最早是一个哲学上的概念，从哲学的范畴来说，Ontology 是客观存在的一个系统的解释或说明，关心的是客观现实的抽象本质。在人工智能界，最早给出 Ontology 定义的是 Neches 等人，文献[18]将 Ontology 定义为“给出构成相关领域词汇的基本术语和关系，以及利用这些术语和关系构成的规定这些词汇外延的规则的定义”。1993 年，Gruber [18]给出了 Ontology 的一个最为流行的定义，即“Ontology 是概念模型的明确的规范说明”。后来，Borst 在此基础上，给出了 Ontology 的另外一种定义[19]：“Ontology 是共享概念模型的形式化规范说明”。Studer 等对上述两个定义进行了深入的研究，认为 Ontology 是共享概念模型的明确的形式化规范说明。这包含 4 层含义[20]：概念模型(conceptualization)、明确(explicit)、形式化(formal)和共享(share)。

“概念模型”指通过抽象出客观世界中一些现象(Phenomenon) 的相关概念而得到的模型。概念模型所表现的含义独立于具体的环境状态。

“明确”指所使用的概念及使用这些概念的约束都有明确的定义。

“形式化”指 Ontology 是计算机可读的（即能被计算机处理）。

“共享”指 Ontology 中体现的是共同认可的知识，反映的是相关领域中公认的概念集，即 Ontology 针对的是团体而非个体的共识。

Ontology 的目标是捕获相关领域的知识，提供对该领域知识的共同理解，确定该领域内共同认可的词汇，并从不同层次的形式化模式上给出这些词汇（术语）和词汇间相互关系的明确定义。

2.1.2 本体的作用

本体对领域知识进行了一种表述，统一了领域内的术语和概念。从而增加了知识共享、知识重用的程度。Uschold 等人在《本体：原理、方

法和应用》一文中总结了本体的作用，即交流(communication)、互操作(inter-operability)和系统工程(systems engineering)。

交流：主要为人与人或组织与组织之间的交流提供共同的词汇。通过公认的词汇集（本体）来减少因对概念理解的不一致，以及个人表达习惯不同而带来的交流上的障碍。

互操作：将本体作为中间表示在不同的建模方法、规范、语言和软件工具之间进行翻译和映射，以实现它们的互操作和集成。

系统工程：本体可以为系统工程带来规范描述(specification)、重用(re-usability)、可靠性(reliability)等好处。本体有助于确定系统的需求和规范；所得的本体可以在其它的工程中共享和重用；非形式化的本体可以帮助设计者检查系统的设计，而形式化的本体，使自动的一致性检查成为可能，从而提高了软件的可靠性。

2.1.3 Ontology 的描述语言

目前在具体应用中 Ontology 的表示方式主要有 4 类[21]：

- 非形式化语言
- 半非形式化语言
- 半形式化语言
- 形式化语言

其形式化的程度越高越有利于机器的自动处理。

可以用自然语言来描述 Ontology，也可以用框架、语义网络或逻辑语言来描述。

自上个世纪 90 年代以来，一些基于 AI 的本体实现语言陆续被提出，如 KIF，Ontolingua，CycL，Loom，OCML，FLogic。后来，随着 Web 的发展，又出现了一系列基于 Web 的本体语言，也叫做本体标记语言，如 SHOE，XOL，RDF，RDF-S，OIL，DAML，DAML+OIL，OWL。

⑴ SHOE[22]

SHOE (Simple HTML Ontology Extensions)作为 HTML 的扩展，是马里兰大学开发的。它是基于框架和规则的。它使用不同于 HTML 的一些标记，使得可以在 HTML 文档中插入本体。当 XML 产生并成为 Web 上

交换信息的标准后，SHOE 的语法被修改为基于 XML。目前，马里兰大学已经停止研究 SHOE，它们有关本体的研究项目开始使用 OWL 和 DAML+OIL 作为本体的描述语言。

⑵ XOL[23]

XOL (Ontology Exchange Language)是 SRI International 的人工智能中心(AIC)开发的。它是一种简单通用的定义本体的方法。其目的是在不同的数据库、本体开发工具、或者其它应用程序之间交换本体。XOL 设计之初是为生物信息学领域本体的交换，但是它可以应用于各种领域。

⑶ RDF，RDF-S[24]

XML 是一种语义语言，也是资源描述语言，但是 XML 对于资源关系描述的能力非常贫乏，因此为了加强资料的处理性，例如资料交换，W3C 提出可资源描述框架 RDF(Resource Description Framework)，其主要的目的是为元数据在 WEB 上的各种应用提供一个基础的架构，从而实现元数据的交换。

⑷ DAML-OIL[25]

DAML-OIL (DARPA Agent Markup plus Ontology Inference Layer)是由欧洲与美洲的联合会所发展出来的一种语义语言。DAML+OIL 的能力是建立在 XML 及 RDF 的基础上，这些 XML 的应用帮助提供了语义网页的初步的能力。用 DAML+OIL 所表示的语言适合用来建立语义网页，DAML+OIL 的语法延伸了 RDF(S)也就是从 RDF 的三元组改变为 n-triple，所以，DAML+OIL 的语言更为复杂。另外 DAML+OIL 也提供一些规则描述资源间的关系及限制(constraints)，所以 DAML+OIL 所表示的文件不仅机器可以理解并能够对内容做推理(reasoning)。

DAML-OIL 延伸了 XML 与 RDF 的优势，并且弥补了 RDF 所不能描述的关系，以下是 DAML+ OIL 与 RDF 的比较：

① 描述能力更优于 RDF，属性可描述属性，类别的关系可以做 equivalence 以及 disjointness 运算，所以可表达更多更复杂的关系。

② Layered approach XML=>RDF(S)=>DAML+OIL.XML 与 RDF(S)语法可用于 DAML+OIL，它们是相容的关系。

领域内资源的语义，来自与表示知识的概念：

① DAML 十 OIL 可视为描述逻辑。

② 可描述 Ontology。

③ 具有逻辑的规则及推理的能力。

④ DAML+OIL 对资源描述以及关系表达十分的强大，所描述的资源也十分的丰富。

⑸ OWL[26] 。

OWL（Web Ontology Language）是 W3C 推荐的本体描述语言的标准，位于 W3C 绘制的本体语言栈的栈顶（参见图 2）。它是为了在 WWW 上发布和共享本体而提供的语义标记语言。OWL 是在 DAML+OIL 的基础上发展起来的，作为 RDF(S)的扩展，目的是提供更多的元语以支持更加丰富的语义表达，并更好的支持推理。我们将在下一节中对 OWL 语言进行详细介绍。

2.1.4 本体开发工具

目前 W3C 以及有关研究机构已经推出了以语义 Web 中知识本体为核心的相关标准、编辑工具和开发软件包，为开展面向知识内容的语义检索提供了支持。其中典型的编辑工具有美国斯坦福大学的 Protégé软件，德国 Karlsruhe 大学的 KAON 工具中的 OI2Modeler，德国 Ontobroker 项目组的 OntoEdit 和英国曼切斯特大学和阿姆斯丹公立大学的 OILED 等。此外，还推出了应用于本体中查询和推理的开发包，如 HP 公司的面向 J2EE 平台的 Jena 开发包、德国 Karlsruhe 大学的 RDF API 等，面向 Windows 和. Net 平台的 RDF. net、Vcsoft 开发包等。

Jena 作为惠普实验室开发的开放资源，是实现语义网应用系统(包括语义检索系统)的有力工具，也是研究和实现语义检索的基本工具之一。Jena2 支持基于 RDFS 和 OWL 等语义推理。Jena 支持一种语义网查询语言:RDQL。Jena2 还拥有一种表现层接口是 RDF Web API。它能提供 Web 客户端查询 RDF 图。这种基于 Web 查询的数据获取方式当然也可以为系统和应用程序员提供接口。方便程序员对本体的各种操作。

2.2 本体描述语言 OWL

2.2.1 OWL 语言的简介

OWL 全称 Web Ontology Language，是 W3C 推荐的语义互联网中本体描述语言的标准。它是从欧美一些研究机构的一种结合性的描述语言 Daml+Oil 发展起来的，其中 Daml 来自美国的提案 DAML-Ont，Oil 来自欧洲的一种本体描述语言。在 W3C 提出的本体语言栈中，OWL 处于最上层，如图 2 所示。

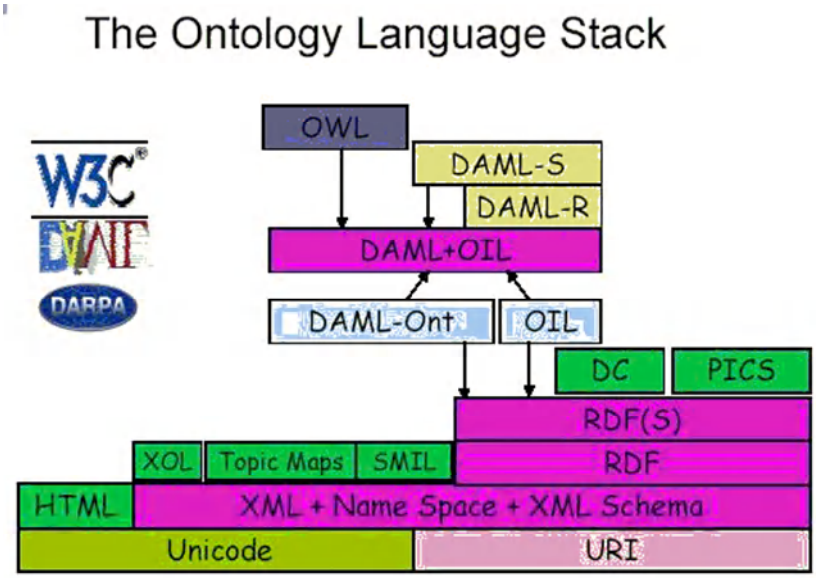


图 2 OWL 在语义网中的层次

针对不同的需求 OWL 有三个子语言，描述列表如下：

⑴子语言：OWL lite

描述：用于提供给那些只需要一个分类层次和简单属性约束的用户。

例子：支持基数(cardinality)，只允许基数为 0 或 1。

⑵子语言：OWL dl

描述：支持那些需要在推理系统上进行最大程度表达的用户，这里的推理系统能够保证计算完全性(computational completeness，即所有地结论都能够保证被计算出来)和可决定性(decidability，即所有的计算都在有限的时间内完成)。它包括了 OWL 语言的所有约束，但是可以被仅仅置于特定的约束下。

例子：当一个类可以是多个类的一个子类时，它被约束不能是另外一个类的实例。

⑶子语言：OWL full

描述：支持那些需要在没有计算保证的语法自由的 rdf 上进行最大程度表达的用户。它允许在一个 ontology 在预定义的(RDF、OWL)词汇表上增加词汇，从而任何推理软件均不能支持 OWL full 的所有 feature。

例子：一个类可以被同时表达为许多个体的一个集合以及这个集合中的本体的描述与推理一个个体。

这三种子语言之间的关系是：

每个合法的 OWL lite 都是一个合法的 OWL dl；

每个合法的 OWL dl 都是一个合法的 OWL full；

每个有效的 OWL lite 结论都是一个有效的 OWL dl 结论；

每个有效的 OWL dl 结论都是一个有效的 OWL full 结论。

用户在选择使用哪种语言时的主要考虑是：选择 OWL lite 还是 OWL dl 主要取决于用户需要整个语言在多大程度上给出了约束的可表达性；选择 OWL dl 还是 OWL full 主要取决于用户在多大程度上需要 rdf 的元模型机制(如定义类型的类型以及为类型赋予属性)； 在使用 OWL full 而不是 OWL dl 时，推理的支持不可预测，因为目前还没有完全的 OWL full 的实现。

这三种子语言与 rdf 的关系是：

OWL full 可以看成是 rdf 的扩展；

OWL lite 和 OWL full 可以看成是一个约束化的 rdf 的扩展；所有的 OWL 文档(lite，dl，full)都是一个 rdf 文档；所有的 rdf 文档都是一个 OWL full 文档；只有一些 rdf 文档是一个合法的 OWL lite 和 OWL dl 文档。

OWL 与 RDFS 相比，前者对类和属性的表示能力进行了扩展，增加了类的关系(如 disjointness)、集的势(如 exactlyone)、等价性、更丰富的属性类型、新的属性特征(如属性的对称性)、枚举类等。OWL 由三个子语言组成: OWL Lite、OWL DL 及 OWL Full。OWL Lite 是为那些仅需要“类层次”及“简单约束”(集的势只允许有 0 或 1)的开发者而提供的，

使用它时，开发过程相对简单。OWL DL 的表达能力比 OWL Lite 要强(增加了 oneOf、hasValue、disJointWith 等算子，及对“集的势”的完全支持等) ，同时使用 OWL DL 表示本体可保证任何推理在有限时间内完成。OWL Full 的表达能力最强，它允许在 Ontology 中增加并使用 OWL 及 RDFS 之外的词汇，因此，不能保证在任何用 OWL Full 表示的 Ontology 上进行任何推理都能在有限时间内完成。实际上，OWL DL 是 OWL Lite 的扩展，而 OWL Full 可看作是 OWL DL 的扩展，表 1 中用*号标注的算子(或公理)是 OWL DL 和 OWL Full 在 OWL Lite 的基础上增加或扩展的算子(公理)。开发者在选择使用哪种语言的时候，应从表达能力和推理能力两个方面综合考虑。

2.2.2 OWL 的 RDF Schema 特性

OWL 与 RDF Schema 有关系的特性如下：

Class(类)：类定义了具有某些共同属性的一些个体。多个类可以用子类关系组织为一个特定的层次结构。一个内置的最一般的类被称为 Thing，该类表示的是所有个体，因此它是所有 OWL 类的父类。另外一个内置的特殊类 Nothing，它没有任何实例，因此是任意 OWL 类的子类。

Rdfs:subClassOf(子类)：可以给出一个或多个关于“一个类是另一类的子类”的陈述来创建一个类层次结构(Class hierarchies)。例如，可以声明类 Person(人)是类 Mammal(哺乳动物)的子类。推理机就可以据此推出:如果某个个体是一个“人”，那么它也是一个“哺乳动物”。

rdf:Property(属性)：属性表述个体之间或者从个体到数值的关系。分为对象属性(ObjectProperty)和数据属性(DatatypeProperty)两类。

Rdfs:subPropertyOf(子属性)：通过给出一个或多个陈述声明“某属性是另外一个或多个属性的子属性”可建立属性层次。例如，可以声明 hasSibling(有兄弟姐妹)是 hasRelative(有亲戚)的子属性。据此，推理机可以推出:如果一个个体以 hasSibling 属性与另一个体相关联，那么它也一定可以与这个个体以 hasRelative 属性相关联。

rdfs:domain(定义域)：一个属性的定义域用来约束该属性可以适用的个体。如果一个个体通过一个属性和另一个体关联，并且该属性的定义

域是某个类，那么该个体必然属于这个类。rdfs:domain 是全局限制，该限制是在属性上声明的，而不是只有当这个属性应用于某个类时才声明。

表 1　OWL 构造算子和公理对应的 DL 语法的表示

OWL 中的类构造子与 DL 的关系

构造子	DL 语法
intersectionOf*	$C_1 \bigcap \ldots \bigcap C_2$
unionOf*	$C_1 \bigcup \ldots \bigcup C_n$
complementOf*	$\neg C$
oneof*	$\{x_1, \ldots\ldots, x_n\}$
allValuesFrom	$\forall P.C$
someValuesFrom	$\exists P.C$
maxCardinality*	$\leq nP$
minCardinality*	$\geq nP$

OWL 中的公理与 DL 的关系

公理	DL 语法
subClassOf*	$C_1 \subseteq C_2$
equivalentClass*	$C_1 \equiv C_3$
disjointWith*	$C_1 \subseteq \neg C_2$
sameIndividualAs	$\{x_1\} \equiv \{x_2\}$
differentFrom	$\{x_1\} \subseteq \neg \{x_2\}$
subPropertyOf	$P_1 \subseteq P_2$
equivalentProperty	$P_1 \equiv P_2$
inverseOf	$P_1 \equiv P_{2-}$
transtiveProperty	$P_+ \subseteq P$
functionalProperty	$T \sqsubseteq \leq 1P$
InverseFunctionalProperty	$T \sqsubseteq \leq 1P_-$

rdfs:range(值域)：一个属性的值域用来限制哪些个体可以成为属性的值。如果一个个体以一个属性和另一个体关联，并且该属性的值域是

一个类，那么另外那个个体必然属于此类。值域也是全局限制。

Individual(个体)：个体是类的实例，个体之间可以用属性相互关联。

2.2.3 OWL 中的等价和不等价

下面描述 OWL 语言中关于类、属性及个体间的等价性和不等价性。

owl:equivalentClass：当两个类被声明为等价时，就是声明它们有相同的实例。等价性可以用来创建同义类。例如，类 Car 可以被说成是类 Automobile 的等价类。据此，推理机可以推出：任何 Car 的实例都是 Automobile 的实例，反之也成立。

owl:equivalentProperty：两个属性也可以被声明为等价。相互等价的属性将一个个体关联到同一组其它个体。它也可以被用来创建同义属性。例如，hasLeader 可以说成是 hasHead 的等价属性(owl:equivalentProperty)。据此，推理机能够推出：如果 X 通过属性 hasLeader 与 Y 关联，那么 X 也通过属性 hasHead 与 Y 关联。推理机还能推出：hasLeader 是 hasHead 的子属性，hasHead 同时也是 hasLeader 的子属性。

owl:sameAs：两个个体可以声明为相同。这个构词用来创建一系列指向同一个个体的名字。

owl:differentFrom：一个个体可以声明为和其它个体不同。例如，在声明 Frank 是与 Deborah 不同的个体之后声明 Frank 和 Deborah 同为一个函数型（functional 即属性值最多一个）属性的值时，就会产生矛盾。我们可以看到，在使用如 OWL (RDF)等语言时，由于这些语言没有假设个体有且只有一个名字，所以明确声明个体是不同的显得很重要。例如，没有其它附加信息的话，推理机不能够推出 Deborah 和 Frank 指的是不同的个体。

ow1:AllDifferent：在一个 owl:AllDifferent 陈述中，我们可以指出一定数量的个体两两不同。例如，可以用 ow1:AllDifferent 声明 Frank，Deborah，Jim 两两不同。

ow1:AllDifferent 与 owl:differentFrom 不同之处在于，owl:AllDifferent 同时强调了 Deborah 与 Jim 也是不同的（不仅仅是 Frank 不同于 Deborah 和 Frank 不同于 Jim）。

ow1:AllDifferent 的重要性体现在表达一个集合中的对象两两互不相同且建模者有意强调这些个体的独立性时。它常和 owl:distinctMembers 一起使用来声明列表中的成员都是独特的且两两不同。

2.2.4 OWL 的属性特征、属性约束和受限基数

OWL 中有一些特定的标识符用以提供关于属性及其值的信息。主要的几个有：

owl:inverseOf：一个属性可以声明为另一个属性的逆属性。如果 P1 被声明为 P2 的逆属性，那么如果 X 通过 P2 关联到 Y，则 Y 通过 P1 关联到 X。

owl:TransitiveProperty：属性可以声明为传递性的。如果(x，y)是传递属性 P 的一个实例，(y，z)也是传递属性 P 的一个实例，那么(x，z)是传递属性 P 的一个实例。OWLLite 和 OWL DL 给出了关于传递属性的一个边界条件：传递属性和它的父属性(superproperty)不能有基数(owl:cardinality)限制。如果没有这个边界条件，OWL Lite 和 OWL DL 都将成为不可判定语言。

owl:SymmetricProperty：属性可以被声明为是对称的。如果(x，y)是对称属性 P 的一个实例，那么(y，x)也是它的一个实例。

owl:FunctionalProperty：属性可以被声明为只有唯一值。即一个属性如果被声明为函数型属性(owl:FunctionalProperty)，那么对于每个个体，属性最多只有一个值(也可能对某些个体没有值)。

owl:InverseFunctionalProperty：如果一个属性被声明为具反函数性质的(inverse functional)，那么它的逆属性是函数型的，也就是说该属性的逆属性对任何一个个体来说最多只有一个值。这个特征也被称为单值(unambiguous)属性。例如，hasIDNumber(身份证号)就可以声明为反函数型。对任意个体(其类型是身份证号)，该属性的逆属性的值最多一个。由此任何人的身份号是逆属性。据此，推理机可推出：没有两个不同的人(类 Person 的实例)能有相同的身份证号。另外，还可以推断出:如果两个人(Person 的实例)有相同的身份证号，那么这两个实例表示同一个个体。

OWL 允许对属性添加约束以使其能够适用于一些特定类的实例。这

些类型在 owl:Restriction 中使用。元素 owl:onProperty 用于指示被约束的属性。

owl:allValuesFrom：约束 ow1:a11ValuesFrom 是对属性适用于某个类时声明的。它的意思是这个属性用于这个类时有一个局部性值域约束。因此，如果一个类的实例通过这个属性和另一个体关联，那么另外那个个体就能够被判定为是该局部值域约束类的一个实例。但是推理机并不能从属性的 owl:allValuesFrom 约束推出属性至少有一个值。

owl:someValuesFrom：约束 owl:someValuesFrom 是对属性适用于某个类时声明的。该约束指这个属性至少有一个值是属于某个类型的。

OWL 包含了一个有一定使用限制的基数约束。OWL 的基数约束被称为局部约束，因为它们是对一个属性应用于某特定类时的声明。也就是说:这类约束应用于那个类的实例时就会给出属性的基数信息。OWL Lite 的基数约束限制在只允许基数值为 0 或 1(不像在 OWL DL 和 OWL Full 中那样允许基数值为任意数目)。

owl:maxCardinality：最大基数是对属性适用于某个类时声明的。如果属性适用于某个类时的 owl:maxCardinality 为 1，则这个类的任意实例都通过这个属性和至多一个个体关联。有约束 owl:maxCardinality 为 1 的属性有时也叫做函数型(functional)或者唯一值(unique)属性。仅仅从最大基数为 1 的约束，推理机不能推出其最小基数也为 1。某个属性的最大基数值为 0 表示该属性没有值。

owl:minCardinality：最小基数是对属性应用于某个类时声明的。如果就某个类而言，声明一个属性的 owl:minCardinality 最小基数为 1，则这个类的任何一个实例都通过该属性至少和一个个体相关。这也是一种用来表达对于某一个类的每个实例，这个属性必须至少有一个值的办法。OWL Lite 中，最小基数的值只允许为 0 或 l。0 表示的意思是对于某个类而言这个属性是可选的。

owl:cardinality：基数用于简便地表示属性适用某一个类时同时具有约束“ owl:minCardinality”为。并且“owl:maxCardinality”为 0 或者同时具有约束“owl:minCardinality”为 1 和“owl:maxCardinality”为 1 的情况。

2.2.5 OWL 类的并、交、补

owL 包含了一个有使用限制的交集构词。

owl:unionOf：类的并集。

owl:intersectionOf：类的交集。OWL 允许在具名类和约束中的交集。例如，类 EmployedPerson（雇员）可定义为类 Person 和类 EmployedThings（被雇佣者）的交集，其中类 EmployedThings 可定义为拥有 hasEmployer（有雇主）属性并且该属性有约束 minCardinality 为 1。推理机可以据此推出：任意一个 EmployedPerson 的实例都具有至少一个雇主。

owl:completmentOf：类的补集。

2.3 本体开发语言 JENA

2.3.1 JENA 语言的简介

Jena 是来自于惠普实验室语义网研究项目的开放资源[36,37]，是用于创建语义网应用系统的 Java 框架结构，它为 RDF、RDFS、OWL 提供了一个程序开发环境[38]。具体包括用于对 RDF 文件和模型进行处理的 RDF API，用于对 RDF、RDFS、OWL 文件（基于 XML 语法）进行解析的解析器，RDF 模型的持续性存储方案，用于检索过程推理的基于规则的推理机子系统，用于对 Ontology 进行处理和操作的 Ontology 子系统，用于信息搜索的 RDQL 查询语言。Jena 的这些组成部分在解决语义网环境下语义检索中各司其职，起到重要的作用如图 2 所示。

在图 2 中，RDF/XML 文档是信息资源的原始存储和标引格式，它可以通过 RDF/XML 解析器和 RDF API 转变成 RDF Model 存储在计算机的内存中，也可以通过 RDF 模型的持续化存储方案和 RDF API 将 RDF Model 存储在数据库中并可以通过 RDF API 随时调用，这有利于数据量较大的 RDF Model 的存储；RDF Model 可以直接用于信息检索，但是通常情况下要结合推理机子系统和 Ontology 子系统生成具有语义推理能力的 InfModel 或者 OntModel 从而实现语义检索的目标；通过 RDQL 对 Model 的检索结果经过一定的处理之后便可以与用户进行交互。

Jena 是一个基于 Java 语言的先进的语义网开发工具。它的第一个版

本是 Jena1，发布于 2000 年，已经被下载了 10000 多次。Jena2 修改了前一个版本的内部架构，又提供了一些新功能，于 2003 年 8 月发布。现在已经被下载 13000 多次。

语义网推荐规范中的本体描述语言的核心是 RDF 图(Graph)，这是全球通用的数据结构。一个 RDF 图是由一组三元组(Predicate，Subject，Object)组成，P 是(Subject，Object)的一个二元谓词关系。Jena2 同样是围绕 Graph 作为核心接口，然后来构建其它的组件。

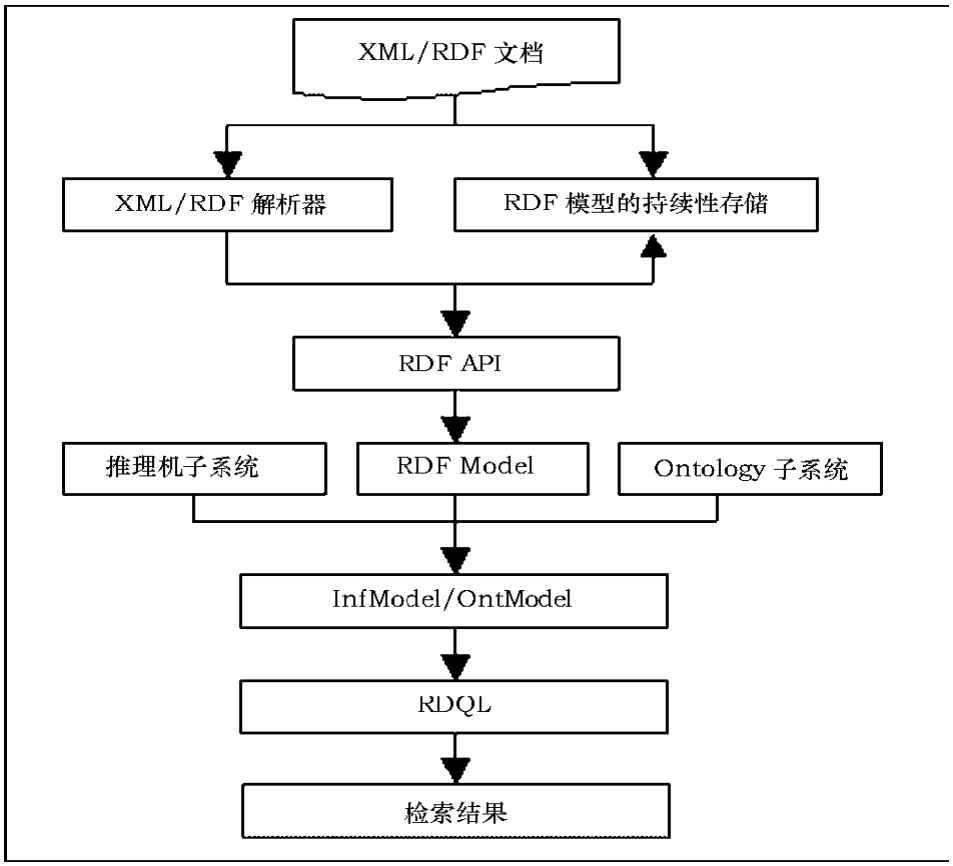


图 2　Jena 各组成部分在语义检索中的作用

Jena1 的主要贡献是为处理 RDF 图提供了丰富的用于 Model 类的 API。围绕着这些 API， Jena1 提供了大量的工具，包括对模型(Model)的多种方式的输入输出 I/O(RDF/XML，N3，N-triple)，RDF 查询语言 RDQL。使用这些 API，用户可以选择将 RDF 图存储在内存中或者是持久性存储（文件或数据库方式）。Jena1 还提供了附加的用于处理 DAML+OIL 数据的 API，但是 Jena1 并不支持 OWL。

后来，在很多用户的反馈中建议 Jena 更好的整合对 DAML+OIL 和对 RDF 的支持，以求达到更强的功能，例如存储 DAML 数据模型到数据库中。对于 Jena1 来说实现起来很困难，因为 Model 类的 API 已经很多了，无法再继续实现更多的 API。

为了解决这些问题，Jena2 采用了更好的系统架构。以求达到两个关键目标：

对于应用开发者可以提供多种灵活的表现 RDF 图的方案。这需要允许用户可以使用更高层接口或使用底层接口的不同方式访问处理 RDF 图数据。

提供一种 RDF 图的最简单的视图方式——三元组方式，主要便于系统级程序开发人员处理数据。这对基于 RDFS 和 OWL 推理非常有用。

第一点实际上是建立在第二点的基础上的。两个关键目标都是为系统程序员提供扩展的功能点。RDF 图的表现方式不仅是现有的 Model API 而且是新的 Ontology API（用于 OWL，DAML+OIL，RDFS 等）的基础。而在 RDF Graph 的表现层应该允许新三元组源的开发，包括存储在数据库或内存的三元组以及运算处理过程中动态产生的一些三元组（比如推理过程中产生的数据源）。

Jena2 支持基于 RDFS 和 OWL 等语义推理。Jena 支持一种语义网查询语言:RDQL。Jena2 还拥有一种表现层接口是 RDF Web API。它能提供 Web 客户端查询 RDF 图。这种基于 Web 查询的数据获取方式当然也可以为系统和应用程序员提供接口。这种方式有可能是 Jena 的以后的发展方向。

2.3.2 RDF 应用程序接口

RDF API 的主要功能包括创建和读、写 RDF 模型，操纵和检索 RDF 模型[39]。

(1)创建和读、写 RDF 模型：Jena 具有一系列的方法可以创建一个 RDF 模型并可以将其内容写入到 RDF 文件中，也可以把 RDF 文件中的信息读取到模型中，如表 2 所示，当然相关的 RDF 文件都要遵循 XML 语法和格式规则。

表 2 创建和读、写 RDF 模型

```
//创建一个空的 RDF 模型
Modelmodelname = ModelFactory.createDefaultModel()
//将模型中的信息写入到 RDF 文件
modelname.write(system.out)
//将 RDF 文件中的信息读取到模型中
modelname.read(new InputStreamReader(inputstreamname)，””)
```

(2)操纵和检索 RDF 模型：Jena 提供一系列的方法用于对 RDF 模型中的数据进行操纵和检索。在操纵 RDF 模型时，首先可以根据 Resource 的 UR I（Universal Resource Identifier）地址从模型中提取一个 Resource 对象，然后可以利用 Resource 对象提供的接口来操纵其中的相关内容，例如检索符合一定条件的特定值；使用 RDF API 对 RDF 模型只能进行比较粗糙和简单的检索，更强大的检索功能要借助于 RDQL 查询语言，可以使用一定的 Jena API 方法在 RDF 模型中检索出符合条件的信息。如表 3 所示。

表 3 在 RDF 模型中检索信息

```
//根据 Resource 的 UR I 返回 Resource 对象
Resource resourcename = model.getresource(URI)
//利用 Resource 对象的接口列出符合条件的信息(一个例子)
String stringname = resourcename.getProperty(propertyname).getString ()
//使用一般的方法检索 RDF 模型
ResIterator iteratorname = model.listSubjectsWithProperty (propertyname)
//使用 Selector 对象以三元组匹配的形式检索 RDF 模型
Selector selectorname = new SimpleSelector(Subject，Predicate，Object)
//然后可以使用 liststatement 方法输出相关结果
StmtIterator interatorname = model.listStatements(selectorname)
```

2.3.3 RDF 模型的持续性存储问题

无论是 RDF、RDFS 还是 OWL，其描述结构都是主（Subject）、谓（Predicate）、宾（Object）的三元组结构，除了在内存中存储数据模型

之外，对数据量较大的 Data 或者 Ontology 数据可以通过 RDF 提供的数据库引擎将其存储在数据库表中，用户可以根据需要将这些数据从数据库中检索出来从而创建 RDF 模型以供进一步使用和处理。目前，Jena 数据库引擎支持 MySQL、Oracle 和 PostgreSQL 等数据库，支持 Linux 和 Windows XP 操作系统。对数据库的具体操作如表 4 所示。

表 4 JENA 对数据库的操作

```
//加载数据库 JDBC 驱动类
Class.forName(className)；
//创建一个数据连接
Conn = new DBConnection (DB URL，DB-USER，DB PASSWD，DB)；
//配置本体描述规范，采用 OWL_DL 语言，不用推理机
OntModelSpec spec=On tModelSpec. OWL-DL_MEM；
spec.setModelMaker(ModelFactory.createModelRDBMaker(conn))；
//创建 db_gghz 本体库(空)
Model base = maker.createModel(”db_gghz”)，
OntModel m = ModelFactory.createontologyModel(spec，base)；
//读入本体库文件
URL u1；
u1 = C1assLoader.getSystemResource(ontoFile)；
m.read(u1.toString()，“RDF/XML-ABBREV“)；
//获取本体库 db_gghz
Model m = ModelFactory.createModelRDBMaker(conn.openModel
(“db_gghz “)；
```

语义检索是基于概念及其概念之间的关系进行的语义层面的检索，其关键在于概念之间的推理。Jena 提供基于规则的推理机（如 RDFS Reasoner、OWL Reasoner 等），它包含了一般的推理功能，此外用户还可以根据需要自定义推理规则，也可以注册使用第三方推理引擎。如图 3 所示，推理机的工作原理是：推理机注册机制根据基本 RDF 三元组描述(信息资源)和 Ontology（可选）创建出推理机，由此推理机可以生成包含推理机制的模型对象（Inference Graph，InfGraph），在 Jena 中，图（Graph）

也被称为模型（Model），而表现形式为模型界面（ModelInterface），然后可以使用 Model API 和 Ontology API 对此模型进行操作和处理，从而实现语义层面的信息检索。

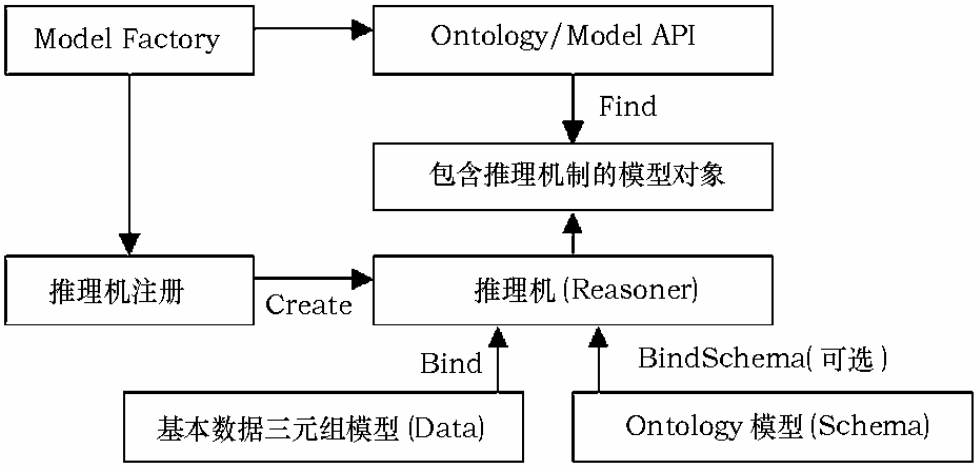


图 3　推理机的工作机制

2.3.4　推理子系统

在使用“推理机注册”创建推理机的时候，有三种方式：

和 OWL Reasoner，同时支持 DAML。这些推理机中的推理规则都是基于一般用途的，以 OWL Reasoner 为例，具体的使用过程如表 5 所示。

表 5　推理机的使用(以 OWL Reasoner 为例)

```
//获取 Ontology 数据
Model schema = ModelLoader.loadModel(SchemaFileURI)
//获取信息资源数据
Model data = ModelLoader.loadModel(DataFileURI)
//创建空的 OWL 注册机
Reasoner reasoner = ReasonerRegistry.getOWLReasoner()
//向注册机中绑定 Ontology 数据
reasoner = reasoner.bindSchema(schema)
//向注册机中绑定信息资源数据
InfModel infmodel = ModelFactory.createInfModel(reasoner，data)
```

(2)使用基于自定义规则的推理机。推理机的内部是根据一定的触发机制通过对推理规则的解释从而实现推理的效果。具体的触发机制包括向前链引擎、向后链引擎和混合式规则引擎，向前链引擎和向后链引擎可以彼此独立使用，或者向前链引擎可以初始化向后链引擎；对于推理规则，用户可以根据需要定制自己的规则，然后根据自定义的规则可以创建特定的推理机，实际上，Jena 自带的推理规则可以在$\etc 下的相关文件中找到。自定义推理规则的推理机的创建如表 6 所示。

表 6　自定义规则推理机的创建和使用

```
//自定义推理规则
String rules = ” [r1: (?c eg:concatFirst ?p)，(?c eg:concatSecond ?q) -> ”
+ ” [r1b: (?x ?c ?y) <- (?x ?p ?z) (?z ?q ?y)]] ”
//根据自定义推理规则创建对应的推理机
Reasoner reasoner = new GenericRuleReasoner(Rule.parseRules (rules))
//根据自定义的推理机创建包含推理关系的数据模型
InfModel inf = ModelFactory.createInfModel(reasoner，rawData)
```

(3)使用第三方推理机。除了 Jena 自带的推理机机制之外还有其它可以应用在语义网中的推理机，这些推理机可以与 Jena 进行集成，所以对于 Jena 而言，我们称之为第三方推理机。

2.3.5 RDQL 查询语言

RDQL 是 RDF 的查询语言。虽然 RDQL 还不是正是的标准，但已由 RDF 框架广泛执行。RDQL 允许简明地表达复杂的查询，查询引擎执行访问数据模型的繁重工作。RDQL 的语法表面上类似 SQL 的语法，它的一些概念对已经使用过关系数据库查询的人来说将比较熟悉。就像 SQL 语句从关系数据库的数据库表中查询数据一样，RDQL 从 RDF Model（包括 InfModel 和 OntModel 等）中检索数据，检索的结果存储在一定的数据结构中可供调用[28]，如表 7 所示，将其与用户的检索界面实现联接和集成，则可解决语义检索应用系统与用户交互的问题。

表 7 用 RDQL 实现信息检索示例

```
//构建查询语句
String querystring = " Select ……"
//创建查询对象
Query queryname = new Query(querystring)
//设定检索目标 Model
Queryname. setSource(modelname)
//创建查询执行对象
QueryExecution qename = new QueryEngine(queryname)
//查询执行并存储结果
QueryResults resultsname = qename.exec()
```

2.3.6 Jena 推理规则的构造语法

用户一方面可以利用上面的结构实现本体中的推理，另一方面，如果用户对 Java 很熟悉，则可以自己开发推理规则。规则的语法格式如下：

```
Rule := bare-rule
        or [bare-rule]
        or [ruleName :bare-rule]
bare-rule := term，...term->hterm，...hterm      //向前推理
        or term，...term<-term，...term          //向后推理
hterm := term
        or [bare-rule]
term := (node，node，node)                       //三元组模式
        or (node，node，functor)                 //扩展三元组模式
        or builtin (node，...node)               //调用处理元语
funclor  := functorName (node，...node)          //结构化的文字表述
node := uri-ref                                  //例如 http://w3c.com/eg
        or prefix:localname                      //例如 rdf:type
        or ? varname                             //变量名
        or 'aliteral'                            //字符串
```

or number //例如 7 或 14

2.4 小 结

本章详细介绍了 RDF 的蕴涵语义；OWL 语言的构造算子和公理；编译工具 JENA 的使用方法。这些是本文研究工作所涉及到的相关基础理论和工具，为以后对 OWL 算子的推理规则研究和算法的实现打下基础。

第三章 本体推理规则的研究

本章基于对本体推理、RDF 蕴涵规则和 OWL 语言算子的研究，找出了 OWL 中一些基本约束类和属性之间的推理关系，详细给出了一些重点算子的推理规则关系。

3.1 本体推理的介绍

3.1.1 本体推理的基本工作

描述逻辑提供的推理能力是有限的[36]，比如它不提供关系间的组合推理，而对应用推理而言，多数规则表述的是众所周知的含义，因此价值不大。规则列表之中有一些表达 subclass，subProperty 的传递性以及对 property 的约束如：range，domain 等的一些语义蕴涵规则对应用推理而言，具有较高的价值。

例如：在本体语言蕴含的规则中，有这样一条规则“子类的实例也是父类的实例”。

(owl:subClassOf ?child ?parent
Rdf:type ?instance ?child)
=>
(Rdf:type ?instance ?parent)

如上所示，当在知识库中出现“孩子是父亲的子类”并且“存在一个孩子的实例”时，这条规则就被触发，执行的相应结果就是申明了“这个实例也是父亲的实例”的事实，并添加到知识库中。

再比如：

假设我们已经给出了 hasFather 和 hasGrandFather 的定义。

<Person rdf:ID=”TOM”>
<hasFather rdf:resource=”#Jack”>
</Person>

<Person rdf:ID=”Jack”>

<hasFather rdf : resource=”#Mike”>

</Person>

却无法推出

<Person rdf : ID=”TOM”>

<hasGrandFather rdf : resource=”#Mike”>

</Person>

即推出 TOM 的爷爷是 Mike，而这却是很普通的一种推理。

因此，有必要对本体语言进行语义规则的扩充。达到对本体的补充和完善。对规则添加的研究可以结合描述逻辑和数据库触发器规则。

现阶段，对本体的推理研究主要集中在以下五方面：

- 本体中的冲突检测
- 本体中的表达优化
- 本体的融合
- 获取本体中的知识
- 验证知识的正确性

3.1.2 常见的推理问题

从一个本体中获取隐性知识一般分为下面五个方面：

(1) 类（概念）——实例关系推理：给定知识库 K，C 是 K 中的一个类(概念)，I 是 K 中的一个个体，可对以下类与实例的关系进行推理：判断一个个体是否是 C 的一个实例；判断在 K 中 C 的所有实例；判断在 K 中个体 i 是哪些类的实例；判断两个实例之间的关系或判断与某个实例有特定关系的实例。

(2) 类（概念）的关系推理：给定类 C 和 D，判断它们之间的关系（子类、成员、部分等）。

(3) 在类的体系结构中进行推理：给定类 C，返回在 K 中 C 的所有或相关的超类。或者在 K 中 C 的所有或相关的子类。

(4) 类的满足性推理：给定一个类 C，判断是否 C 在 K 是可满足的（一致的）。

(5) 基于属性的推理：属性与类（或者实例）有相似的推理，包括：属性—实例关系，属性包含，属性体系结构和属性可满足性等。

3.1.3 本体论知识表示的推理

知识推理的一个基本内容就是由给定的知识获得隐性的知识。推理有多方面的应用:对于知识库的建立者，推理的主要作用是检测冲突和优化表达；对于知识库的使用者，推理的作用主要在于获得知识库中的知识和运用知识库中的知识进行问题的解决。

目前知识推理的研究热点，描述逻辑（Description Logic），目的是在表示能力与推理复杂度之间取得平衡。描述逻辑是基于命题逻辑（Propositional Logics）和一阶谓词逻辑（First order Logic）上发展而来。因为命题逻辑的概念只能作为原子命题存在，表达力明显不足。而一阶谓词逻辑又由于表达力过强而导致相关的推理算法过于复杂，无法有效地控制推理时间，它们都不很适合用于本体。1988 年，Schmidt-Schau.B 提出了 ALC （Attributive Concept Description Language）。这套语言包括概念定义，二元关系定义，概念的交、并、补，关系约束等等。在 ALC 中有一套成熟的 Tableau 算法可以实现其概念的可满足性推理而且复杂度较低，同时，概念的包含推理可以化为可满足性来实现。Tableau 算法的思想就是把可满足的概念集合通过一系列推理规则不断细化、扩展，直到发现冲突。如果直到概念被完全扩展仍然没有不可避免的冲突，就可以得到满足概念集合的一个解，从而证明概念集合是可以满足的。另外，用一些新的语法把 ALC 的表达力增强产生了 ALCN，ALCQ， ALCF 等等，这些语言构成了 ALC 系列。

描述逻辑知识库：TBox

TBox(Terminology Box)是一个描述领域的结构的公理集。它包括：

定义公理 $(definition\ axioms)A \equiv C$：$woman \equiv human \cap female$；

$motherOFdaughters \equiv woman \cap (\geq 1has-child.\top) \cap \forall has-child.woman$

包含公理

$(inclusion\ axioms)C_1 \subseteq C_2$：$\exists has-\deg ree.PhD \subseteq \exists has-\deg ree.PhL$;

$binaryTree \subseteq node \cap (\leq 2edge.\top) \cap \forall edge.binaryTree$

说一个解释 I 满足 $A \equiv C$，当且仅当 $AI \equiv CI$；

满足 $C_1 \subseteq C_2$，当且仅当 $C_1 I \subseteq C_2 I$；

满足 TBox T 当且仅当满足每个公理 $x \in T$。

描述逻辑知识库：ABox

ABox(Assertional Box)是一个描述关于具体个体事实的公理集。

概念断言(concept assertional)a:C;Peter:student;cat∩∃offspring.⊥

角色断言(role assertional)R(a，b):has-friend(Peter，Mary)。

解释 I=(ΔI，·I)映射每个个体 a 到一个元素 $a_I \in \Delta I$。

一个解释 I 满足 a：C 当且仅当 a_I:CI；

满足 R(a，b)当且仅当(a_I，b_I)⊆RI；

满足 ABox A 当且仅当满足每个断言 a∈A。

本体中大部分知识不是显式说明的而是隐性的。而且，考虑到不能因本体过于庞大造成效率的低下，本体应用也不能把所有隐性的知识显式的表示出来。因此，获取本体中隐性的知识就是推理机的基本任务。知识获取的效率由本体的复杂度和其组织的优化程度决定，具体实现可能会需要使用一些产生式规则来配合搜索算法。这项工作包括两方面：

1. 获取 Tbox 中类的关系和属性的关系，具体来说有获得类的子类集(就是包含推理)、属性的子属性集、属性的特征、属性的定义域和值域等等。它更接近于概念的查询，因此推理的可靠性要求较高，不能出现概念错误。

2. 获取 Abox 中的知识，具体来说是获得一个类的所有实例、实例的属性、获得满足条件的所有实例等等，也就是实例的检索和信息的浏览，这项工作对于推理的速度和效率要求均较高，而所获得信息的完整性成为了次要需求。

在引入本文的本体的描述性定义之前，我们先来介绍一些用于我们定义本体规范概念的基本知识：

一般来说，一个 Ontology 由概念(concepts)，关系(relations)，函数(functions)，公理(axioms)，实例(instances)等五种元素组成。

(1) 本体中的概念是广义上的概念，它除了可以是一般意义上的概念以外，也可以是任务、功能、行为、策略、推理过程等等。本体中的这

些概念通常构成一个分类层次。

(2) 本体中的关系表示概念之间的一类关联，典型的二元关联如子类关系形成概念类的层次结构，一般情况下用 R :C1 ×C2 ×…×Cn 表示概念类 C1 ，C2 ， …，Cn 之间存在 n 元关系 R。

(3) 函数是一种特殊的关系，其中的第 n 个元素相对于前 n - 1 个元素是唯一的，一般情况下，函数用 F :C1 ×C2 ×…×Cn - 1 →Cn 表示。

(4) 公理用于表示一些永真式。更具体地说，在许多领域中，函数之间或关联之间也存在着关联或约束。

(5) 实例是指属于某概念类的基本元素，即某概念类所指的具体实体，特定领域的所有实例构成领域概念类在该领域中的指称域。

本体中的基本关系有如下四种：

part-of 概念之间部分与整体的关系；

kind-of 概念之间的继承关系，类父子类关系；

instance-of 概念的实例和概念之间的关系，类对象和类之间的关系；

attribute-of 某个概念是另一概念的属性。

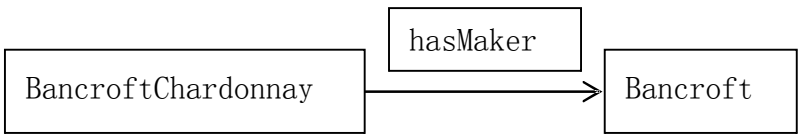
本文中用于推理的几种本体关系的属性：

1．反转性：也为逆关系(inverse) 。例如：

```
<owl:ObjectProperty rdf:ID="hasMaker">
  <rdf:type rdf:resource="&owl；FunctionalProperty" />
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="producesWine">
  <owl:inverseOf rdf:resource="#hasMaker" />
</owl:ObjectProperty>
```

这里，属性 producesWine 和 hasMaker 就有逆关系。

```
<Chardonnay rdf:ID="BancroftChardonnay">
  <hasMaker rdf:resource="#Bancroft"/>
</Chardonnay>
```



由此，我们可以推出 Bancroft 生产出(producesWine) BancroftChardonnay，表示成 OWL 语言如下：

```
<Winery rdf:ID="Bancroft">
  <producesWine rdf:resource="#BancroftChardonnay"/>
</Winery >
```

2．传递关系(transitivity)

当概念 A 和概念 B 有关系 R 的同时，概念 B 和概念 A 有关系 R－1，则称关系 R 具有逆属性，或称关系 R 是可逆关系。例如：

```
<owl:ObjectProperty rdf:ID="locatedIn">
  <rdf:type rdf:resource="&owl；TransitiveProperty"/>
  <rdfs:domain rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:range rdf:resource="#Region"/>
</owl:ObjectProperty>
```

这里，定义了属性 locatedIn 具有传递性。

```
<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="#CaliforniaRegion" />
</Region>
<Region rdf:ID="CaliforniaRegion">
  <locatedIn rdf:resource="#USRegion" />
</Region>
```

由此，我们可以推出 SantaCruzMountainsRegion 是位于(locatedIn)USRegion 里的。

用 OWL 语言表示如下：

```
<Region rdf:ID="SantaCruzMountainsRegion">
  <locatedIn rdf:resource="#USRegion"/>
</Region>
```

3．继承关系(kind-of)

如概念 C 和 D，记 C’＝{x|x 是 C 的实例}，D’={x|x 是 D 的实例}，如果对任意的 x 属于 D’，x 都属于 C’，则称 C 为 D 的父概念，记作 C=Parent(D)；称 D 为 C 的子概念，记作 D=Child(C)。例如：

<owl:Class rdf:ID="Food"/>

<owl:Class rdf:ID="Wine">

<rdfs:subClassOf rdf:resource="#Food"/>

</owl:Class>

<Wine rdf:ID="FormanChardonnay"/>

属性 subClassOf 是典型的继承关系关键词。Wine 是 Food 的子类，而 FormanChardonnay 是 Wine 类的一个实例。由继承关系，我们可以推出 FormanChardonnay 也是 Food 类的一个实例。

用 OWL 语言表示如下：

<owl:Thing rdf:about="#FormanChardonnay ">

<rdf:type rdf:resource="#Food"/>

</owl:Thing>

4．部分关系(part-of)

概念之间的部分和整体之间的关系，如果概念 C 是概念 D 的一部分，记作 C＝Part(D)。例如：

<owl:Class rdf:ID="Food"/>

<owl:Class rdf:ID="Wine">

<rdfs:subClassOf rdf:resource="#Food"/>

</owl:Class>

这里 Wine 类就是 Food 类的一部分。Wine 类和 Food 类就是部分与整体的关系。

5．实例关系(instance-of)

表达概念的实例和概念之间的关系，E 是概念 C 的实例，记作 E＝Instance(C)。例如：

<owl:Thing rdf:ID="CentralCoastRegion" />

<owl:Thing rdf:about="#CentralCoastRegion">

<rdf:type rdf:resource="#Region"/>

</owl:Thing>

这里 CentralCoastRegion 就是 Region 的一个实例。

6．属性关系(attribute-of)表达某个概念是另一个概念的属性，C 是 D

的属性，记作 C＝Attribute(D)。例如：

```
<owl:ObjectProperty rdf:ID="hasWineDescriptor">
  <rdfs:domain rdf:resource="#Wine" />
  <rdfs:range   rdf:resource="#WineDescriptor" />
</owl:ObjectProperty>
```

这里属性 hasWineDescriptor 的定义域是 Wine，也就是说 hasWineDescriptor 是 Wine 的一个属性。

本体还有一些其它属性，这里不一一列举。

对 RDF 规则的扩展：

蕴涵：在逻辑中，蕴涵是从语义的角度来研究如何从一些在某种解释下为真的语句来推导出其它为真的语句。RDF 推理中也引入了蕴涵的概念。

简单蕴涵：设 S 和 E 是 RDF 图，如果每个满足 S 的简单解释都满足 E，则称 S 简单蕴涵 E。即如果 S 的每一个模型同时也是 E 的模型，称 S 简单蕴涵了 E。

蕴涵规则：模型理论语义本身并没有直接提供一个应用程序操作模型的方式，它只是提供了一种形式化的语义蕴涵定义，还需要定义蕴涵规则让应用程序可以做推理。在 RDF 中，这些规则的基本形式是：如果 RDF 图中包含了某些形式的三元组，则可以根据蕴涵规则在 RDF 图中加入对应的三元组。RDF 语义规范[29]中将蕴涵规则主要分为简单蕴涵规则，RDF 蕴涵规则和 RDFS 蕴涵规则三大类。

表 8，表 9，表 10 分别给出了 RDF 蕴涵规则，RDFS 蕴涵规则和 RDF 基本规则的扩展。

表 8 RDF 蕴涵规则

规则名称	如果 E 包含	则添加
rdf1	u p y	a rdf:type rdf:Property
rdf2	u p l	_:n rdf:type rdf:XMLLiteral _:n 是通过规则 lg 分配给 lll 的空白结点标识符

表 9 RDFS 蕴涵规则

规则名称	如果 E 包含	则添加
rdfs1	u p l	_:nnn rdf:type rdfs:Literal
rdfs2	p rdfs:domain x u p y	u rdf:type x
rdfs3	p rdfs:range x u p v	v rdf:type x
rdfs4a	u p x	u rdf:type rdfs:Resource .
rdfs4b	x p u	u rdf:type rdfs:Resource
rdfs5	a rdfs:subPropertyOf b b rdfs:subPropertyOf c	a rdfs:subPropertyOf c
rdfs6	x p y a rdfs:subPropertyOf b	x b y
rdfs7	x rdf:type rdfs:Class	x rdfs:subClassOf rdfs:Resource
rdfs8	x rdfs:subClassOf y y rdfs:subClassOf z	x rdfs:subClassOf z
rdfs9	x rdfs:subClassOf y a rdf:type x	a rdf:type y
rdfs10	u rdf:type rdfs:Class	u rdfs:subClassOf u
rdfs11	u rdfs:subClassOf v v rdfs:subClassOf x	u rdfs:subClassOf x
rdfs12	u rdf:type rdfs:Container MembershipProperty	u rdfs:subPropertyOf rdfs:member
rdfs13	u rdf:type rdfs:Datatype	u rdfs:subClassOf rdfs:Literal

表 10 RDF 基本规则的扩展

规则名	如果 E 包含	条件	则添加
lg	v p l	$l \in L$	v p bl
gl	v p bl	$l \in L$	v p l
rdf1	v p w		P type Property
rdf2-D	v pl	$l = (s,\ a) \in L + D$	bl type a
rdfs1	v pl	$l \in Lp$	bl type Literal
rdfs2	p domain u v p w	$v\ p\ w$	v type u
rdfs3	p range u v p w	$w \in U \bigcup B$	w type u
rdfs4a	v p w		v type Resource
rdfs4b	v p w	$w \in U \bigcup B$	w type Resource
rdfs5	v subPropertyOf w w subPropertyOf u		v subPropertyOf u
rdfs6	v type Property		v subPropertyOf v
rdfs7x	p subPropertyOf q v p w	$q \in U \bigcup B$	v q w
rdfs8	v type Class		v subClassOf Resource
rdfs9	v subClassOf w u type v		u type w
rdfs10	v type Class		v subClassOf v
rdfs11	v subClassOf w w subClassOf u		v subClassOf u
rdfs12	v type Container-MembershipProperty		v subPropertyOf member
rdfs13	v type Datatype		v subClassOf Literal

3.2 对 RDF 规则的扩展

D* 继承[36,37]的概念概括了关于已给定数据类型的图D对数据类型的包含推理的 RDFS 继承[38]。如果 D 只包含标准数据类型 rdf:XMLLiteral，那么 D*继承完全符合 RDFS 继承。表列举出 D*继承的全部规则。在这个表中类似 rdf:的前缀被忽略，如 URI rdf:type。这些规则由在[8]中定义的关于 RDFS 的 18 个规则组成的，只有 rdf2 和 rdfs7 有些不同。规则 rdfs7x 改正了文献[36] 和文献[37]中的一个错误[37]：它和 rdfs7 的不同在于，它可以产生一般的在断言位置应用普通 RDF 三元组的空白结点的 RDF 三元组，对数据类型的操作，规则 rdf2 替代了更普通的规则 rdf2-D。USE 是由新的空白接点 BL 组成的，BL 又叫代理空白接点，由规则 LG 分配。在规则 rdfs1 中，bl 是由 gl 分配的一个空白结点。

D*的继承是比 D 继承弱的[34]。例如：关于 XML 表数据类型 xsd:boolean，三元组 a p true，a p false ，b type Boolean 由 D 继承推出三元组(a p b)，但是，这不是 D* 继承。

pD* 继承的概念在文献[36，37]中被当作 OWL 继承的变量被介绍。在表 10 中的 18 个 D*继承规则再增加了表 11 中的 23 个 P 继承规则后，完善成了 pD*继承。除在 rdfp15 外，对 someValuesFrom 还有第二中继承规则，叫做 rdf-svx，类似的还有规则 rdfp16 中的 allValuesFrom。这个规则引进了新的空白结点，并且可以被 R 继承增加。在 pD*语义，有一种 P 冲突：它不是 v differentFrom w 和 v sameAs w 两个三元组的并，就是 v disjointWith w，u type v，u type w 三个三元组的并。

3.3 对 OWL 语言中的一些复杂类和限制属性的推理

OWL 语言是在 RDF 上发展起来的，所以研究 RDF 的蕴涵和继承规则对我们研究 OWL 语言的推理性有很大帮助。在第二章中，我们给出了 OWL 的构造算子。每个算子对应着不同的 RDF 语义。

我们先讨论几种 OWL 的构造算子，然后给出其它的 OWL 类和属性的推理规则。

1．owl:sameAs

<Wine rdf:ID="MikesFavoriteWine>

<owl:sameAs rdf:resource="#StGenevieveTexasWhite"/>

</Wine>

在这个例子并没有什么实际意义。所有我们从中能了解到的就是 Mike 喜欢一种便宜的本地酒。sameAs 的一种更加典型的用法是将不同文档中定义的两个个体等同起来，作为统一两个本体的部分。

但这样做带来了一个问题。OWL 中并没有名称唯一这一假定。仅仅名称不同并不意味着这两个名称引用的是不同的个体。

在上面的例子中，我们对两个截然不同的名称作出一致性断言。但是也只有在这种标示的情况下，才可能进行推理。那些可能从函数型属性中得出的含意。假如 hasMaker 是一个函数型属性，那么下面的例子就不一定会产生冲突。

<owl:Thing rdf:about="#BancroftChardonnay">

<hasMaker rdf:resource="#Bancroft"/>

<hasMaker rdf:resource="#Beringer"/>

</owl:Thing>

除非和我们本体中的其它信息发生冲突，不然的话这样的描述是没有冲突的，它说明 Bancroft 和 Beringer 是相同的个体。

用 OWL 语言表示如下：

<Winery rdf:ID="Beringer">

<owl:sameAs rdf:resource="#Beringer"/>

</ Winery>

在本文中，对于给定的本体，我们认为是完全正确的，所以根据上面的关系，我们给出：

如果实例属性 P 有值为 1 的限制属性 minCardinality，切本体中一个实例的 P 属性有 2 个不同的值 T1，T2，则推出：T1 和 T2 具有 sameAs 属性。

2．owl:intersectionOf

<owl:Class rdf:ID="WhiteWine">

<owl:intersectionOf rdf:parseType="Collection">

```
        <owl:Class rdf:about="#Wine" />
        <owl:Restriction>
          <owl:onProperty rdf:resource="#hasColor" />
          <owl:hasValue rdf:resource="#White" />
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
```

使用集合操作构造的类与我们目前所看到的所有语法中的定义类似。类的成员完全是通过集合操作来说明的。上面的语句说明 WhiteWine 恰好是类 Wine 与所有颜色是白色的事物的集合的交集。这就意味着如果某一事物是白色的并且是葡萄酒，那么它就是 WhiteWine 的实例。如果没有这样的定义我们只能知道白葡萄酒是葡萄酒酒并且是白色的，但是反过来就不是这样了。这是对个体进行分类的强有力工具。

```
    <owl:Class rdf:about="#Burgundy">
      <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#Wine" />
        <owl:Restriction>
          <owl:onProperty rdf:resource="#locatedIn" />
          <owl:hasValue rdf:resource="#BourgogneRegion" />
        </owl:Restriction>
      </owl:intersectionOf>
    </owl:Class>
```

在这里我们定义了 Burgundy 类。这个类恰好包含了那些至少有一个 locatedIn 关系，而同时这一关系又要联系到 Bourgogne 地区的葡萄酒。当然我们也可以声明一个新的类 ThingsFromBourgogneRegion，并且将该类作为 owl:intersectionOf 结构中的类使用。但既然 ThingsFromBourgogneRegion 不再有其它用处，上面的声明就显得更加简短、清晰，并且这一声明不需要我们努力想一个新的名字出来。

```
    <owl:Class rdf:ID="WhiteBurgundy">
      <owl:intersectionOf rdf:parseType="Collection">
```

```
        <owl:Class rdf:about="#Burgundy" />
        <owl:Class rdf:about="#WhiteWine" />
      </owl:intersectionOf>
    </owl:Class>
```

因为，WhiteBurgundy 是 Burgundy 和 WhiteWine 的交集，所以 WhiteBurgundy 必须具有 Burgundy 和 WhiteWine 的属性。所以，很容易的推出 WhiteBurgundy 具有 hasColor 属性，切其值为 White。和具有 locatedIn 属性，切其值为 BourgogneRegion。

根据上面的关系，我们给出：

如果实例 I 是类 C1 和 C2 的交集。则 I 具有 C1 和 C2 的属性。

3．someValuesFrom 和 allValuesFrom

我们已经发现了一种限制组成属性的元素的类型的方法。到现在为止，我们所讲述的机制都是全局的(global)，因为这些机制都会应用到属性的所有实例。而接下来要讲述的两个属性限制机制，allValuesFrom 与 someValuesFrom，则是 局部的(local)，它们仅仅在包含它们的类的定义中起作用。

owl:allValuesFrom 属性限制要求：对于每一个有指定属性实例的类实例，该属性的值必须是由 owl:allValuesFrom 从句指定的类的成员。

```
  <owl:Class rdf:ID="Wine">
    <rdfs:subClassOf rdf:resource="&food；PotableLiquid" />
    ...
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty rdf:resource="#hasMaker" />
        <owl:allValuesFrom rdf:resource="#Winery" />
      </owl:Restriction>
    </rdfs:subClassOf>
    ...
  </owl:Class>
```

Wine 的制造商必须是 Winery。allValuesFrom 限制仅仅应用在 Wine

的 hasMaker 属性上。Cheese 的制造商并不受这一局部限制的约束。

owl:someValuesFrom 限制与之相似。在上个例子中，如果我们用 owl:someValuesFrom 替换 owl:allValuesFrom，那就意味着至少有一个 Wine 类实例的 hasMaker 属性是指向一个 Winery 类的个体的。

```
<owl:Class rdf:ID="Wine">
  <rdfs:subClassOf rdf:resource="&food；PotableLiquid" />
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:onProperty rdf:resource="#hasMaker" />
      <owl:someValuesFrom rdf:resource="#Winery" />
    </owl:Restriction>
  </rdfs:subClassOf>
  ...
</owl:Class>
```

这两种限制形式间的不同就是全称量词与存在量词间的不同。如表 11 所示。

表 11 allValuesFrom 和 someValuesFrom 的含义

关系	含意
allValuesFrom	对于所有的葡萄酒，如果它们有制造商，那么所有的制造商都是酿酒厂。
someValuesFrom	对于所有的葡萄酒，它们中至少有一个的制造商是酿酒厂。

前者并不要求一种葡萄酒一定要有一个制造商。如果它确实有一个或多个制造商，那么这些制造商必须全部都是酿酒厂。后者要求至少有一个制造商是酿酒厂，但是可以存在不是酿酒厂的制造商。

```
<Wine rdf:ID="SelaksIceWine">
<hasMaker   rdf:resource="#Selaks"/>
</Wine>
```

因为，SelaksIceWine 是一种 wine，wine 的 hasMaker 属性都来自 Winery，而 SelaksIceWine 的 hasMaker 是 Selaks。所以，我们很容易的

推出 Selaks 是 Winery 的一个实例。

上面我们通过 3 个简单例子，给出了 OWL 语言的几个属性的简单推理，下面我们给出其它一些 OWL 类和属性的推理规则，具体推理规则如表 12 所示。

表 12 一些 OWL 类和属性的推理规则

规则名	如果 E 包含	条件	则添加
rdfp1	p type FunctionalProperty u p v u p w	v∈U∪B	v sameAs w
rdfp2	p type Inverse FunctionalProperty u p w v p w		u sameAs v
rdfp3	p type SymmetricProperty v p w	w∈U∪B	w p v
rdfp4	p type TransitiveProperty u p v v p w		u p w
rdfp5a	v p w		v sameAs v
rdfp5b	v p w	w∈U∪B	w sameAs w
rdfp6	v sameAs w	w∈U∪B	w sameAs v
rdfp7	u sameAs v v sameAs w		u sameAs w
rdfp8ax	p inverseOf q v p w	w，q∈U∪B	w q v
rdfp8bx	p inverseOf q v q w	w∈U∪B	w p v

规则名	如果 E 包含	条件	则添加
rdfp9	v type Class v sameAs w		v subClassOf w
rdfp10	p type Property p sameAs q		p subPropertyOf q
rdfp11	u p v u sameAs u’ v sameAs v’	u’∈U∪B	u’ p v’
rdfp12a	v equivalentClass w		v subClassOf w
rdfp12b	v equivalentClass w	w∈U∪B	w subClassOf v
rdfp12c	v subClassOf w w subClassOf v		v equivalentClass w
rdfp13a	v equivalentProperty w		v subPropertyOf w
rdfp13b	v equivalentProperty w	w∈U∪B	w subPropertyOf v
rdfp13c	v subPropertyOf w w subPropertyOf v		v equivalentProperty w
rdfp14a	v hasValue w v onProperty p u p w		u type v
rdfp14bx	v hasValue w v onProperty p u type v	p∈U∪B	u p w
rdfp15	v someValuesFrom w v onProperty p u p x x type w		u type v

规则名	如果 E 包含	条件	则添加
rdfp16a	v allValuesFrom w v onProperty p u type v u p x	x∈U∪B	x type w
Rdfp16b	v allValuesFrom w v onProperty p p range u		w subtype u
rdfp17	u disjointWith v u'subClassOf u v'subClassOf v		u' disjointWith v'
rdfp18a	u complementOf v		v complementOf u
rdfp18b	u complementOf v		v disjointWith u

其中，U 表示 URI references 的集合，B 表示空白结点。

3.4 小 结

在这一章中，我们分析了 RDF 蕴涵规则，并研究了 OWL 语言中的一些公理，属性和实例的基本算子与 RDF 蕴涵规则之间的关系，针对 OWL 中这些基本约束类和属性，找出它们之间的推理关系，给出了详细的推理规则。

第四章 ORBO 算法

根据前述知识表示方法，本文使用的本体由 OWL 文档构成的，而 OWL 文档作为 XML 文档，是一种树形状态的文件，树的每个结点是一个 XML 结点)，结点与结点之间的关系为父子关系或者是兄弟关系。因此，每个 OWL 文档可以表示为一个树形结构的状态空间。这样一来，一些传统的对接结点的推理算法就可以应用到本体的推理中来。如，Baarder[36]等人提出的 Structural Subsumption 算法；Ian Horrocks 等人提出的 Role absorption [37]算法等。2006 年，Li 等人[17]提出了基于 RDF 和 PD*语义的推理算法，该算法实现了对本体的推理，但推理时间复杂度还比较高。通过分析 Li 的算法的不足，本文提出了 ORBO 算法，对 Li 的算法进行改进，降低了时间复杂度。

4.1 Li 基于 RDF 和 PD*语义的推理算法

4.1.1 Li 算法介绍

2006 年， Li 等人 [17] 提出了一种基于 RDF 和 PD*语义的正向推理算法 PD*算法。该算法的主要思想是：

1．初始化所有规则作为触发器。

2．读入 RDF 图 G 和所有自定义三元组。

3．开始循环。

4．对每条规则，判断它是否在最后一个迭代中被触发。如果它的前提与最后迭代中产生的三元组相匹配，那么就把这个规则运用到图 G 中，并记录下由它产生的规则。

5．当没有新的三元组产生时，循环结束。

这个算法是以分析了 RDF 蕴涵规则为前提，结合了著名的 sesame 算法和 PD*语义的基础上给出的。

在这个算法中，当一条确定的规则被适用于图 G 的时候，也就是说，首先我们查找在最后一次迭代中产生的新三元组作为一个规则的三元组匹配前提，如果成功，则尝试在 G 中找到其它前提，如果所有的前提都

能被匹配，这一条规则的结论被推出而且加到图 G 中。举例来说，当规则 rdfs2 触发，我们将首先查找在最后的迭代中得到的三元组，它至少和一个三元组的前提匹配。如果成功，则查找图 G 中匹配三元组的另一个前提。以这二个前提匹配的三元组将推出结论三元组，所有的这样的三元组对都会被找到，而且推导出相应的结论。最后，被 rdfs2 触发的规则被记录用于下个迭代。

该算法是一个迭带算法，在迭带中判断推理规则是否在最后的迭带中被触发。我们可以看出，这个算法是用结点和推理规则的前提进行匹配，如果结点和推理规则的前提之一匹配，则查找和这条规则前提匹配的其他结点，如果全部存在，则推出结论。

4.1.2 Li 算法的不足

该算法是一个迭带算法，在迭带中判断推理规则是否在最后的迭带中被触发。众所周知，迭带算法都需要以时间复杂度为代价。而且，该算法要求每个结点对每条的每个关系进行判断，只要当有结点和推理规则的一条关系匹配，则根据规则的前提和其他结点进行匹配，这会浪费很多时间和资源，无法达到时间的优化；另外，该算法要对结点和每条推理规则中的每条推理关系都进行匹配。我们考虑，如果一个推理规则的第一条推理关系不被满足，则这个推理规则就不会被满足。

针对上面提到的两个问题，我们考虑从结点出发，判断推理规则中第一条推理关系的前提是否满足，如果满足，则继续判断该条规则中以后的推理关系前提是否满足，这样不用对每条规则的每条推理关系都和结点进行匹配。对于新产生的结点，我们继续用 PD*算法的思想，这样，可以不用对整个结点集进行重新判断，节省了时间，提高了效率，降低时间复杂度。

4.2 ORBO 算法

4.2.1 算法的基本思想

基于以上的分析，我们提出了 ORBO 算法，对上面提出的不足加以解决，算法的基本思想如下：

1．初始化所有规则作为触发器。

2．读入 RDF 图 G 和所有自定义三元组。

3．将图 G 和自定义三元组转化为结点形式，并放入队列 L 中。

4．对 L 中的每个结点，依次判断它是否和规则集中的第一个规则的第一条关系匹配。

如果是，则对判断该结点在该条推理规则的该条推理关系下的子结点继续判断，如果全部完全匹配，则推出结论，并判断该结论结点是否已经在 L 中存在，如果没有，则将新产生的结点放到一个新队列 L1 中；如果不全部匹配，则继续判断是否和规则集中的下一条规则的第一条关系匹配。

如果不是，则判断下一条推理规则。

5．如此依次匹配结点队列中的每个结点，直到结点集 L 中没有结点为止。

6．对 L1 中的结点运用 PD*算法。

下面给出 ORBO 算法的 ADL 语言描述。

4.2.2 算法 ORBO 的 ADL 描述

算法 ORBO(G. P)

//G 表示 RDF 图和所有自定义三元组。P 表示推理规则集。D 是 RDF 图 G 中的一个三元组结点。O、A 和 V 分别是三元组结点中的 object 项、attribute 项和 value 项。D 表示为(O A V)。P[i][j]表示推理规则集中第 i 个推理规则的第 j 个推理关系。T(D，P[i][j])判断 D(O A V)是否和 P[i][j]匹配。DP[i,j]表示 D 经过规则 P[i][j]得到的三元组结点。n 表示推理规则集中，推理规则的条数。m[i]表示第 i 条推理规则有 m 条推理关系。L(k)表示队列 L 中的第 k 个元素，即第 k 个三元组结点。

C1(D，P[i])[对第 i 条推理规则的 j 条推理关系进行判断]

```
j <- 1.
WHILE j≤m[i] DO
(
j <- j+1.
D <- DP[i,j].
```

```
IF T(D, P[i][j]) THEN
ELSE END
)
```

R(i).//R(i)表示根据第 i 条推理规则生成新的三元组结点。

L1(R(i)).// L1(R(i))表示把三元组结点 R 放入队列 L1 中。

C2(D)[对第 i 条推理规则进行判断]

```
i <- 1.
WHILE i≤n DO
(
j <- 1.
IF T(D,P[i][j]) THEN
(
C1(D,P[i][j]).
i <- i+1.
)
ELSE i <- i+1.
)
```

C3(G)[对图 G 的推理算法]

```
LG(G).//LG(G)表示将图 G 的结点压入队列 L 中。
k <- 1.
WHILE P1 ≠ null DO  // P1 表示指向队列 L 的当前元素的指针。
(
C2(L(k)).
k <- k+1.
)
PD(L1).■
```

算法 ORBO 的流程如图 4 所示。

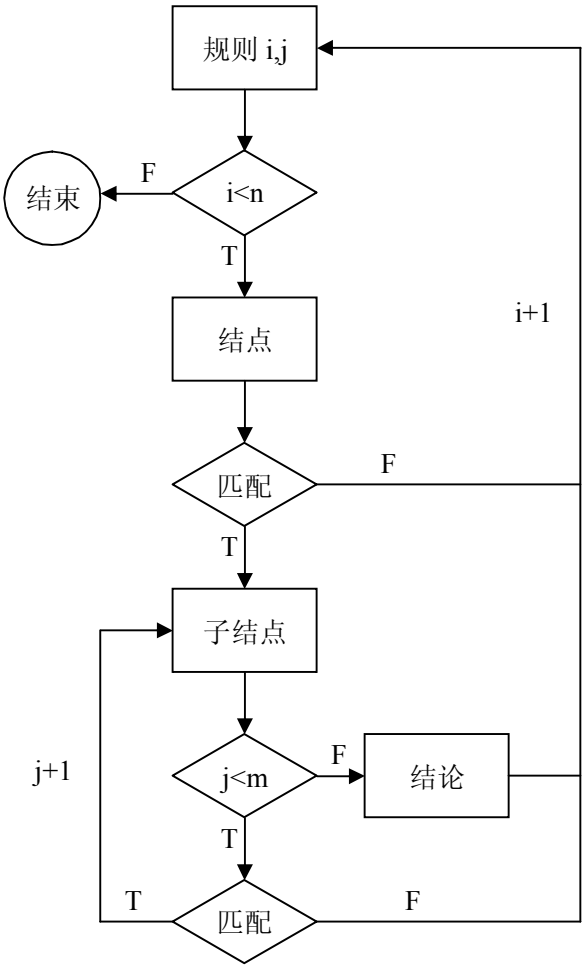


图 4 算法 ORBO 的流程图

ORBO 算法一定会结束：因为，在前半部分，结点和运用的规则总数是一定的，产生的新结点是有限的，所以，前半部分一定会结束；在后半部分，采用了 PD*算法原有的算法，在文献[17]中已经证明了该算法一定会结束。因此，ORBO 一定会结束。

算法的正确性：在前面已经证明了该算法一定会在有限时间内结束；该算法是基于规则前提的判断，所以，在规则前提不满足的情况下，是无法推导出任何结论的，而当规则前提满足了，那么就一定能推导出结论。例如，一个结点 D1 和推理规则 rdfp15 的第一条推理关系的前提 v someValuesFrom w 匹配，则算法会在结点集中查找和 rdfp15 的其他关系匹配的结点，直到全部匹配，推出结论或存在没有匹配的推理关系，则

结点对该条推理规则判断结束，继续判断下一个规则。

在时间复杂度方面，ORBO 算法采用了 WHILE 循环语句，避免了迭带循环，在对新产生的结点则继续沿用 PD*算法的推理。因此，ORBO 算法的时间复杂度为：$O(KMN+K1^{MN})$，而 PD*算法运用迭带算法时间复杂度为：$O((K+K1)^{MN})$. 其中，K 和 K1 分别表示图 G 中原结点数和新产生的结点数。M 和 N 分别表示规则集中有 M 条规则，每条规则最多有 N 条关系。我们可以很容易的对时间复杂度进行比较，$O(KMN+K1^{MN})<O((K+K1)^{MN})$。所以我们提出的 ORBO 算法要比 PD*算法节省很多时间。

有了 3.2 和 3.3 中的推理规则，和 3.4 中的推理算法，我们就可以对本体进行有效的推理了。

4.3 算法实例

下面给出一段本体的 OWL 代码，对算法的流程进行一下说明：

```
<owl:Class rdf:about="#animal">        //定义animal类
<rdfs:subClassOf>
   <owl:Restriction>
    <owl:onProperty rdf:resource="#eats"/>
    <owl:someValuesFrom>
     <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Thing"/>
    </owl:someValuesFrom>
   </owl:Restriction>
</rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#person">        //定义person类
 <rdfs:subClassOf>
   <owl:Class rdf:about="#animal"/>
 </rdfs:subClassOf>
</owl:Class>

<owl:Class rdf:about="#man">           //定义man类
 <owl:equivalentClass>
   <owl:Class>
    <owl:intersectionOf rdf:parseType="Collection">
```

```
        <owl:Class rdf:about="#person"/>
        <owl:Class rdf:about="#male"/>
        <owl:Class rdf:about="#adult"/>
       </owl:intersectionOf>
     </owl:Class>
    </owl:equivalentClass>
    <rdfs:subClassOf>
     <owl:Restriction>
      <owl:onProperty rdf:resource="#eats"/>
      <owl:allValuesFrom>
       <owl:Class rdf:about="http://www.w3.org/2002/07/owl#Food"/>
      </owl:allValuesFrom>
     </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>

  <owl:ObjectProperty rdf:about="#eats">          //定义eats属性
   <owl:inverseOf rdf:resource="#eaten_by"/>
   <rdfs:domain>
    <owl:Class rdf:about="#animal"/>
   </rdfs:domain>
  </owl:ObjectProperty>
```

我们用前面给出的推理规则集为规则集，应用 ORBO 算法：

为方便起见，我们把要用到的推理规则列到下面(如表 13 所示)。

表 13 规则 rdfp1 和 rdfp15

rdfp1	p type FunctionalProperty u p v u p w	$v \in U \bigcup B$	v sameAs w
rdfp15	v someValuesFrom w v onProperty p u p x x type w		u type v

算法从规则集的第一条推理规则开始判断，当运行第 1 条推理规则时，规则的前提没有被任何结点所触发，而开始对第二条推理规则进行

判断。由于篇幅限制，在这里只对有代表性的规则进行叙述。

当运行到第 15 条规则 rdfp15 时，(animal someValuesFrom #Thing)和 rdfp15 前提 v someValuesFrom w 匹配，因此，算法开始判断 rdfp15 的第二个推理关系 v onProperty p，(animal onProperty eats)和 rdfp15 的第二推理关系也匹配，则继续判断第三个推理关系 u p x，找到一个结点，要求，它的三元组形式是(?x eats ?y)，这里找到了结点(man eats #Thing)，第三个推理关系也得到满足，继续第四条(x type w)，第四条前提意思是要求#Thing 属于#Thing，这里#Thing 明显属于自己，所以继续判断，因为没有后继推理关系，所以该条推理规则得到满足，生成推理结论，(man type animal)。然后添加(man type animal)到本体中就。

4.4 小 结

本章通过分析 Li 等人[17]提出的基于 RDF 和 PD*语义的推理算法的不足，提出了 ORBO 算法，给出了 ORBO 算法的详细描述，并通过实例给出了推理的详细推导过程。

第五章　基于 OWL 语言规则推理的实现

在本章中，我们将根据上一章中的推理规则，实现一个基于 OWL 算子的推理规则集。对给定的本体应用该推理规则集，是把隐性在显式定义和声明中的知识提取出来，生成新的更完善的本体。

5.1　系统框架和实现

5.1.1　实现平台及工具

本演示系统的实现平台为 Win32 平台，开发工具为 HP 公司 JENA 的开发环境，JAVA 编译环境。

5.1.2　系统框架

本系统采用了本体和推理规则分离的方案，所以，在设计系统时，考虑了本体和推理规则集分别读入的原理。以达到本系统可以对多本体和多推理规则集的应用。

本演示系统的流程主要分为以下几个步骤：

1) 读入 OWL 文件和推理规则集文件；
2) 对读入的 OWL 本体应用 ORBO 算法进行推理，推导出本体中的隐性知识，加入到本体中；
3) 将新生成的 OWL 本体文件存储到指定文件。

5.2　演示结果

5.2.1　推理规则集编写

对于本体推理机制，最重要的是起推理规则集，首先我们给出在第三章中讨论的推理规则的规则集文件，它以文本文件形式保存。另外，本设计系统中，因为我们采用本体和推理规则集分离的推理系统，所以，我们可以对推理规则集进行修改。

5.2.2　推理系统演示

1. 为用户提供一个友好界面，如图 5 所示。用户选择读入的本体

(OWL 格式)和所要使用的推理规则集。这里，我们读入 W3C 比较常用的 people+pets.owl 本体，它不是很大，但含盖很多内容，是研究本体推理比较常用的本体；推理规则集采用我们在上面给出的自定义推理规则集。

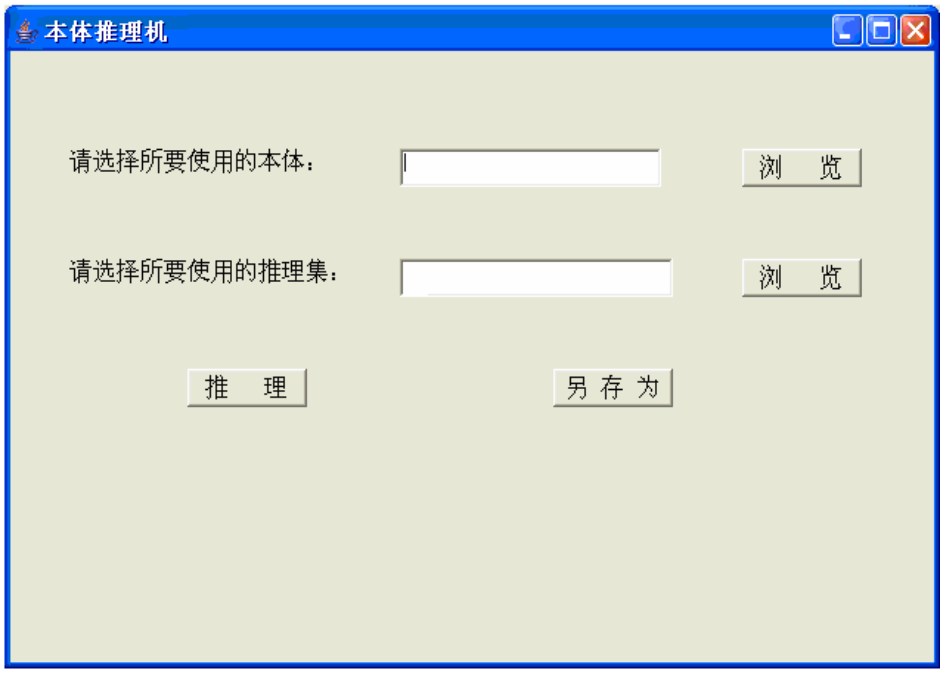


图 5 实现系统的界面

2. 点击推理按钮，应用 ORBO 算法实现对指定本体的推理，如图 6 所示。

3. 将推导后的本体存储到到指定文件(OWL 格式)。

这样，我们就可以将推理后新生成的文件保存为另外的文件名，以防覆盖原文件。

图 6 运行推理规则

5.3 对 比

5.3.1 ORBO 算法和 PD*算法的对比

这里，我们对推理实现系统的算法分别采用 ORBO 算法和 Li 的 PD*算法。

1．读入的本体：我们都使用 W3C 推荐的 people+pets.owl 本体，推理规则集则采用本论文中给出的推理规则集。

输出的结果：除类和属性的顺序及一些描述上的略微差异外，其主要推理结果，基本一致。

运行时间：我们对两个算法分别运行了三次，其运行时间分别为：

①ORBO 算法：4.3 秒，4.4 秒，4.4 秒

②PD*算法：5.1 秒，5.0 秒，5.1 秒

2．读入的本体：我们都使用 W3C 推荐的 wine.owl 本体，推理规则集则依然采用本论文中给出的推理规则集。

输出的结果：除类和属性的顺序及一些描述上的略微差异外，其主要推理结果，基本一致。

运行时间：我们对两种算法分别运行了三次，其运行时间分别为：

①ORBO 算法：7.8 秒，8.0 秒，7.7 秒

②PD*算法：8.8 秒，8.6 秒，8.9 秒

3．读入的本体：我们都使用 W3C 推荐的 wine.owl 本体，推理规则集则采用 JENA 自带的 OWL 规则集 owl.rules。

输出的结果：除类和属性的顺序及一些描述上的略微差异外，其主要推理结果，基本一致。

运行时间：我们对两种算法分别运行了三次，其运行时间分别为：

①ORBO 算法：13.5 秒，13.6 秒，13.6 秒

②PD*算法：16.3 秒，16.7 秒，16.3 秒

对比两种算法的运行时间，如表 13 所示。

表 14 两种算法的运行时间的对比

运行时间	ORBO 算法（秒）	PD*算法（秒）
1	4.3 4.4 4.4	5.1 5.0 5.1
2	7.8 8.0 7.7	8.8 8.6 8.9
3	13.5 13.6 14.0	16.3 16.7 16.3

通过实验数据表明：对不同的本体和不同的推理规则集，两个算法推导出的结果基本一致。在运行的时间上，ORBO 算法都比 PD*算法所用的时间少。当推理规则集的规则越多，ORBO 算法所用的时间比 PD*算法所用的时间减少的越多。

5.3.2 推理前后本体文件的对比

我们给出推理前的本体文件和推理后的本体文件的对比，查看推理的运行结果(部分代码)，如表 14 所示。

表中对比了 2 个类在推理前和推理后的代码，我们可以看到：

表 15 推理前后的对比(部分代码)

推理前：	推理后：
<owl:Class rdf:about="#cat_owner"> <rdfs:label>cat owner</rdfs:label> <rdfs:comment><![CDATA[]]></rdfs:comment> <owl:equivalentClass> <owl:Class> <owl:intersectionOf rdf:parseType="Collection"> <owl:Class rdf:about="#person"/> <owl:Restriction> <owl:onProperty rdf:resource="#has_pet"/> <owl:someValuesFrom> <owl:Class rdf:about="#cat"/> </owl:someValuesFrom> </owl:Restriction> </owl:intersectionOf> </owl:Class> </owl:equivalentClass> </owl:Class>	<owl:Class rdf:ID="cat_owner"> <rdfs:subClassOf rdf:resource="#cat_liker"/> <rdfs:subClassOf> <owl:Class rdf:about="#pet_owner"/> </rdfs:subClassOf> </owl:Class>
<owl:Class rdf:about="#sheep"> <rdfs:label>sheep</rdfs:label> <rdfs:comment><![CDATA[]]></rdfs:comment> <rdfs:subClassOf> <owl:Class rdf:about="#animal"/> </rdfs:subClassOf> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="#eats"/> <owl:allValuesFrom> <owl:Class rdf:about="#grass"/> </owl:allValuesFrom> </owl:Restriction> </rdfs:subClassOf> </owl:Class>	<owl:Class rdf:ID="sheep"> <rdfs:subClassOf rdf:resource="#vegetarian"/> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty> <owl:ObjectProperty rdf:about="#eats"/> </owl:onProperty> <owl:allValuesFrom rdf:resource="#grass"/> </owl:Restriction> </rdfs:subClassOf> </owl:Class>

(1)cat_owner 类在推理前定义为：①如果有 pets 属性，那 pets 属性

至少有一个是来自 cat 类；②是 person。这 2 个条件的交集。在推理后，我们可以很容易的看到 cat_owner 类是 cat_liker 的子类，还是 pet_owner 的子类。这个推理就运用了 rdfp15。

(2) sheep 类，在推理前定义为：①它是 animal 的子类；②它 eat 属性都是来自 grass。在推理后，sheep 类被变成了 vegetarian 的子类。这个推理就运用了 rdfp16。

为方便观看，我们用图来对比推理前后，我们用 protage 打开推理前后的 owl 文件，对比上面的 2 个结果。图 7 为第一个例子的推理前后，图 8 为第二个例子的推理后。

我们将结果和著名的 RACER 推理机推导出的结果进行对比，结果也基本一致。

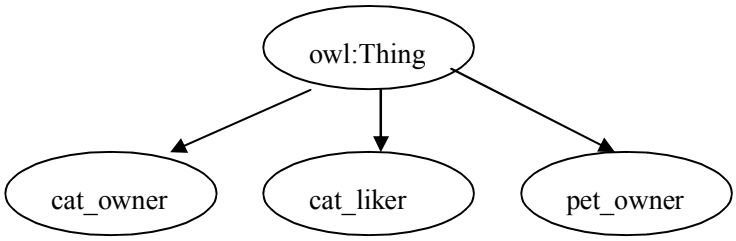


图 7a cat_owner 类推理前的树状图

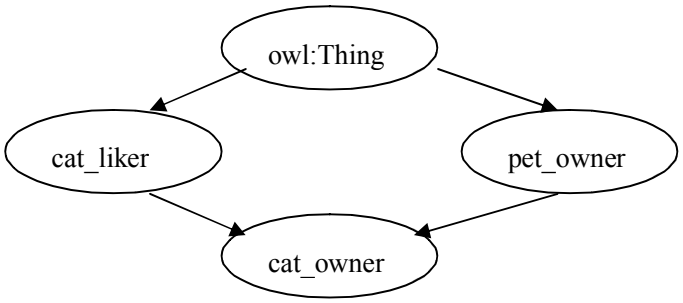


图 7b cat_owner 类推理后的树状图

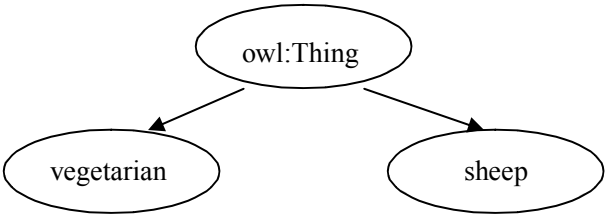


图 8a sheep 类推理前的树状图

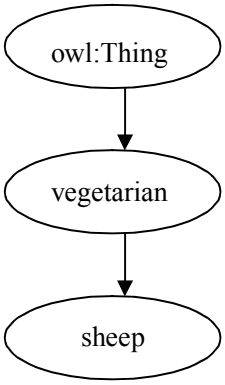


图 8b sheep 类推理后的树状图

5.4 小 结

本章实现了基于 ORBO 算法的演示系统，为用户提供了一个友好界面，实现了对本体在指定规则集的推理；给出了 ORBO 算法和 PD*算法在程序运行时间上的对比；给出了 ORBO 算法推理前后，生成的本体文件的对比。通过实现系统的推理，我们可以获得本体中的隐性知识，为以后本体查询奠定了基础。

第六章 结论与展望

6.1 结 论

近年来，本体理论及其研究越来越受到国内外语义网(Semantic Web)、人工智能(AI)、本体知识库及相关学术界的重视。本文围绕 OWL 语言的构造算子和 RDF 之间的语义，及 OWL 语言所描述的本体推理展开了讨论和研究。

具体工作如下：

1. 对本体和本体描述语言作了简单介绍；
2. 对本文所要用到的本体编程语言 JENA 作了介绍；
3. 通过分析 RDF 蕴涵规则和 OWL 语言的类和属性算子的语义，给出了 OWL 语言中一些类和属性所蕴涵的推理规则；
4. 通过分析 PD*算法，找出 PD*算法的不足，对 PD*算法进行优化，提出了 ORBO 推理算法，给出了 ORBO 算法的详细描述；
5. 设计并实现了一个基于 ORBO 算法演示系统。可用于对 OWL 语言所描述的本体进行推理，得到本体中的隐性知识。

6.2 进一步工作

由于时间的限制和学识上的不足，本文仍尚有许多方面有待扩充和完善。但本体的研究还在起步阶段，所以还有许多问题需要解决。进一步的工作有：

1. 对本文所提出的 OWL 算子之间的推理规则进行更多的讨论，对其所蕴涵的推理规则进行深入研究；
2. 对本文中还没有研究的一些其它 OWL 算子进行分析，并得到它们所蕴涵的推理规则，扩展推理规则集；
3. 对本文所提出的 ORBO 算法进行优化，降低时间复杂度。

参考文献

1. T Berners-Lee, J Hendler, O lassila.The semantic Web. Scientific American. 2001, 284(5):34~43.
2. T Berners-Lee. Semantic Web road map. http://www.w3.org/DesignIssues/Semantic.html.
3. Uschold M, Gruninger M. ONTOLOGIES: Principles,Methods and Applications[J]. Knowledge Engineering Review. 1996:11.
4. 张志雄. Ontology 是什么[J]. 图书情报工作动态. 2004(3):1~6.
5. Jin Song Dong. Software Modeling Techniques and the Semantic Web. Proceedings of the 26th International Conference on Software Engineering. 2004.
6. Franz Baader, Diego Calvanese, et al. The Description Logic Handbook: Theory, Implementation and Applications [M]. U. K.: Cambridge University Press. 2003.
7. Framenet 的官方网站.其中介绍有关于 Framenet 的基本知识. http://www.Icsi.berkeley.edu/~framenet/.
8. GUM.http.//www.darmstadt.gmd.de/publish/komet/gen-um/newUM.Html.
9. SENSUS 语言的官方网站.介绍 SENSUS 语言的使用说明等. http://www.isi.edu/natural-language/resources/sensus.Html.
10. Mikrokmos 的个人网站. 其中包含 Mikrokmos 的论文和发表文章. http://crl.nmsu.edu/Research/Projects/mikro/.
11. Jena的官方网站.介绍Jena的使用说明等.Jena2 Database Interface-Release Notes. http://jena.sourceforge.net/DB/index. html. Oct,2004.
12. Ian Dickinson. Imp lementation experience with the DIG 1.1 specification. http://www. hpl.hp.Com/semweb/publications.html. May,2004.
13. Hanks S, McDermott D. Nonmonotonic logic and temporal projection[J]. Artificial Intelligence 1987 (33):379~412.
14. Yuzhong Qu. A predicate-ordered logic Generation Computer Systems:for knowledge representation on the web[J]. 2004, 20(1):19~26.
15. Dmitry Tsarkov, Ian Horrocks. Efficient Reasoning with Range and Domain

Constraints. In Proc. of the Description Logic Workshop , 2004.

16. Allen J. F Towards a general theory of action and time[J]. Artificial Intelligence，1984(23):123~154.
17. Huiying Li, Yanbing Wang, Yuzhong Qu, Jeff Z. Pan. A Reasoning Algorithm for pD*. ASWC 2006.
18. Dave Reynolds. Jena 2 Inference support. http://jena.sourceforge.net/inference/index.Html. Oct,2004.
19. Seiji Koide, Hideaki Takeda. OWL-Full Reasoning from an Object Oriented Perspective. ASWC 2006.
20. Neches R, Fikes R E, Gruber T R, et al. Enabling Technology for Knowledge Sharing. Artificial Intelligence. 1991,12(3):36~56.
21. Gruber T R. A Translation Approach to Portable Ontology Specifications. Knowledge Acquisition. 1993(5):199~220.
22. Borst W N. Construction of Engineering Ontologies for Knowledge Sharing and Reuse. PhD thesis, University of Twente, Enschede. 1997.
23. Studer R, Benjamins V R, Fensel D. Knowledge Engineering,Principles and Methods. Data and Knowledge Engineering. 1998,25(122):161~197.
24. Uschold M. Building Ontologies:Towards A Unified Methodology. In expert systems. 96, 1996.
25. SHOE 语言的官方网站. http://www.cs.umd.edu/projects/plus/SHOE/.
26. XOLl 语言的官方网站. http://www.ai.sri.comlpkarp/xol/.
27. RDF 语言的官方网站. http://www.w3.org/RDF.
28. DAML+OIL 语言的官方网站. http://co4.inrialpes.fr/xml/daml/.
29. Grigoris Antoniou, Frank van Harmelen. Web Ontology Language:OWL. http://www.w3.org/TR/2004/REC-owl-features-20040210. 2004.
30. Wordnet 语言的官方网站. http://www.cogsci.princeton.edu/~wn/.
31. Jena2 Ontology API. http://jena.sourceforge.net/ontology/index.html. Oct,2004.
32. Andy Seaborne. Jena tutorial, A programmer’s introduction to RDQL. http://Jena.Soureforge.net. Oct,2004.
33. 高琦, 陈华钧. 互联网 Ontology 语言和推理的比较和分析. 计算机应用与软件

[J] 2004,21(10): 73~76.

34. Michael K.Smith, Chris Welty, Deborah L.McGuinness. OWL WebOntology Language Guide. http://www.w3.org/TR/2004/REC-owl-guide-20040210. 2004.
35. P.Hayes (Ed.),RDF Semantics, World Wide Web Consortium Recommendation, http://www.w3.org/TR/2004/REC-rdf-mt-20040210. February 2004.
36. H.J. ter Horst, Extending the RDFS Entailment Lemma, Proceedings 3rd Int. Semantic Web Conference (ISWC2004), Springer LNCS 3298,2004.
37. H.J. ter Horst. Completeness,Decidability and Complexity of Entailment for RDFSchema and a Semantic Extension Involving the OWL Vocabulary,Revised andextended version of [24], Journal of Web Semantics. 2005:79~115.
38. Steffen Staab, et al. Where are the rules?Trends and Controversoes. IEEE Intelligent System. 2003.
39. Baarder, F., W. Nutt: Basic Description Logics, The Description LogicHandbook (eds. Baader et al.). Chap. 2, Cambridge 2003:43~95.
40. Dmitry Tsarkov, Ian Horrocks.Efficient Reasoning with Range and Domain Constraints. In Proc. of the Description Logic Workshop, 2004.

摘　要

2000 年 Tim Berners-Lee 语义网结构的提出使本体的应用越来越受到人们的重视。随着本体的发展，本体描述语言也趋于多样化，每种本体语言都有其本身的构造算子和语义基础。自从 2002 年 OWL（Web Ontology Language）语言成为 W3C 推荐的本体描述语言以来，对 OWL 语言的研究日益重要。因为研究本体描述语言可以使我们更好的了解语言本身的语义，为推理规则的研究做准备。

近年来，本体的推理工作已经受到很大重视，推理研究也越来越成熟，但更多的研究趋向于领域本体的推理，而对 Tbox（公理集）的研究还为数不多。公理集是描述逻辑中的基层部分，它支撑着整个本体，对公理集的推理规则的研究能使我们更好从本体中获取知识。人们可以从本体中获得知识，而一些重要的知识往往都是在本体中隐性给出的。隐性知识是指那些没有一被文字直接表述出来，但隐性在本体内容中或者相关网页提供的一些重要的信息。目前，研究从本体中推导出隐性知识，尤其是从基于 OWL 描述的本体中推导隐性知识，已成为现今本体推理的热点问题之一。

本文通过对 RDF（Resource Description Framework）蕴涵规则和 OWL 语言算子的研究和分析，找出了 OWL 语言中蕴涵的语义；创建了规则集；给出 ORBO 了应用算法，从而实现了从本体中推导出隐性知识。

具体研究内容如下：

1. 对本体研究进行了介绍。

介绍了本体在语义网中的作用和本体的发展现状。

2. 对OWL语言和编译工具JENA进行了详细的介绍。

这些内容的讨论和分析是开展进一步研究的理论基础。简单介绍了 OWL 语言的三个子语言：OWL lite、OWL dl 和 OWL full；OWL 语言的构造算子和公理所对应的描述逻辑关系；介绍了 OWL 语言与 RDF 有关

系的特性，其中包括：详细介绍了 OWL 中，类、属性、及个体之间的等价与不等价关系；介绍了 OWL 语言中的属性特征、属性约束和受限基数和它们的交、并、补。介绍了编译工具 JENA 的使用方法及句法，其中包括：介绍了如何用 JENA 语言实现对本体的建模、读、写等操作；对本体的存储；运用 RDQL 语言对本体查询。

3. 通过对RDF蕴涵规则和属性关系分析，研究了OWL的算子所蕴涵的语义，给出了这些算子的推理规则。

研究本体描述逻辑的基础 RDF，在对 RDF 的类和属性进行讨论的基础上，分析了本体描述逻辑的语义，并从中找出可能的推理规则来，通过对 RDF 语义中所蕴涵的推理规则研究，为本文中基于 OWL 语言的推理规则研究的打下基础；在详细介绍了 OWL 构造算子和描述逻辑之间关系的过程中，讨论了 OWL 算子所蕴涵的语义关系；根据 OWL 中类、属性和实例的关系，分析了它们之间存在的推理关系，并给出了它们之间的推理规则集。

4. 分析了Li提出的基于RDF和PD*语义的算法的不足，提出了ORBO算法，并给出了ORBO算法的ADL描述。

通过分析 Li 提出的基于 RDF 和 PD*语义算法，这个算法是一个迭带算法，该算法中结点会和每个推理规则中的每条推理关系都进行判断。我们发现如果规则的第一条推理关系无法匹配，则整个推理规则是无法匹配的。基于以上的思想，我们提出了 ORBO 算法，并给出了 ORBO 算法的主要思想和 ADL 描述，克服了 PD*算法中的不足，并举例对 ORBO 算法的推导过程做了详细的说明。

5. 设计并实现了基于OWL语言的推理演示系统。

基于本文中提出的推理规则集和 ORBO 算法，实现了一个演示系统，其主要功能包括：实现对指定本体应用推理规则集的推理；将推理后的本体存储到指定文件；提示推理中出现的错误；给出了 ORBO 算法和 PD*算法在运行时间和推理结果的对比，结果表明不论对不同的本体采用相

同的推理规则集，还是对相同本体采用不同的推理规则集，ORBO 算法所用的时间都比 PD*算法少。

RDF 是整个本体描述的基础，本文在研究 RDF 属性关系和蕴涵规则的基础上，通过分析 OWL 语言和 RDF 之间的关系属性及 OWL 语言本身的语义，给出的 OWL 语言推理规则是建立在本体的 TBOX 上的，所以，对于 OWL 语言来说，更具有一般性；给出的 ORBO 算法不是采用传统的对结点推理的方法，而采用判断推理规则前提是否满足的方法，避免了的结点是否重复的判断，从而降低了时间复杂度。

本文的工作对于从本体中推理获得隐性知识及以后的查询工作都具有一定的意义。

Abstract

Since 2000, due to the identification of grammar structure by Time Berners-Lee, people have focused on the application of Ontology. With the development of Ontology, the descriptive language of ontology is going to diversification, and every ontology has its own structure operator and grammar basic. Since 2002, OWL (Web Ontology Language) became the recommended ontology descriptive language of W3C, the research on OWL is more and more important. Because we can know the grammar of language much better by researching on ontology descriptive language, we can prepare for discursion regulation research.

Recently, the discursion of ontology has been focused, and the research of discursion becomes more and more mature, but much research addresses on the discursion domain ontology, not too much research on Tbox. The set of axiom is the basic part of descriptive logic, and it supports the ontology, the research on discursion regulation of axiom can help us obtaining knowledge. People can get the knowledge from ontology, but some of the important knowledge is often get from the recessive ontology. The recessive knowledge is the key information, which is not presented directly but in the ontology content or the relevant webpages. So far, the research on the discursion of recessive knowledge, especially the recessive knowledge deduced from the OWL descriptive ontology, has been the hot point of ontology discursion.

Through the research and analysis on RDF(Resource Description Framework) implication regulation and OWL language operator, this dissertation locates the implication grammar of OWL, creates the regulation set and formulate the applied arithmetic, then implement the recessive

knowledge deduced by ontology.

The specific research content as follows:

1. Introduction of ontology research

The introduction of the ontology function in grammar net and the development status of ontology.

2. The specific introduction of OWL and compile tool JENA

The discussion and analysis are the theory basic for further research. It presents the three sub-language of OWL: OWL lite、OWL dl and OWL full; OWL structure operator and the descriptive logical relation of axiom; the introduction of OWL and RDF characters, which includes: the detailed demonstration of the equivalence and the inequation relation among class, attribute and unit; the introduction of OWL attribute characters, attribute restriction and restricted radices, moreover, their intersection, union and complement. The introduction of compile tool-JENA uses and grammar, which includes the introduction of how-to-use JENA to implement the modeling, reading and writing operations; the save of ontology; query of ontology by RDQL.

3. Through the analysis of the recessive regulation and attributes of RDF, I research on the recessive grammar of OWL operator and give out the deduce regulation of these operators.

Based on the research of RDF class and attributes, the basic RDF of researching ontology descriptive logic has been done, and the deduce regulation has been found. the research of the recessive deduce regulations in RDF supports the research of deduce regulation based on OWL; in the process of introducing the relation between OWL structure operators and descriptive logic, the grammar relation of OWL operators has been discussed; According to the relation of class, attribute and instance, the deduce relation

among them has been analyzed.

4. This dissertation analyze the shortage of the grammar algorithmic based on RDF and PD*, create the algorithmic of ORBO, and the ADL description of ORBO algorithmic.

Through the analysis of RDF and PD* algorithmic created by Li, such algorithmic is a iteration arithmetic, the node of such algorithmic could estimate with every deduce relation of every deduce regulation. We found that if the first deduce relation of regulation cannot be matched, the whole deduce regulation cannot be matched. Based on above ideology, we create ORBO algorithmic, and give out the main ideology of ORBO algorithmic and the description of ADL, solve the shortage problem of PD* algorithmic, and give an example of detailed deducing process for ORBO algorithmic.

5. The design and implementation of the deduce presentation system based on OWL.

Based on the deduce regulation set and ORBO proposed by this dissertation, an presentation system has been implemented, the main functions including: the implementation of deducing the specific ontology deduce regulations; saving the deduced ontology to the specific location; spotting the errors in deducing process; the contrast between the execution time and the deduce result, the contrast result presents no matter the same deduce regulation set used on the different ontology, or the different deduce regulation set used on the same ontology, the time cost of ORBO algorithmic is much less than PD* algorithmic.

RDF is the basic for describing ontology, this dissertation based on the research of RDF attribute relations and recessive regulation, proposes that the OWL deduce regulation base on the TBOX of ontology, through the analysis of the relation attribute between the OWL and RDF and the OWL grammar,

therefore, the OWL has the universality; the proposed ORBO algorithmic do not use the conventional deducing methodology of point-to-point, but use the methodology of estimating the precondition of deducing regulation, which could avoid the duplicate estimate of node and decrease the time complexity.

The work of this dissertation is meaningful for acquiring the recessive knowledge by deducing ontology and querying work in further operations.

致　谢

本论文是在我的导师欧阳继红教授的悉心指导下完成的。三年来，欧阳老师不仅在学业上对我严格要求，给予我许多谆谆教导、鞭策和鼓励，同时在生活中教导我如何做人做事，给予我许多关怀和照顾。从论文选题、查阅资料、开题到讨论、直至论文的定稿审阅，欧阳老师都付出了大量的心血和劳动。欧阳老师严谨的治学态度，正直谦逊的为人品德，一丝不苟的敬业精神，孜孜不倦的科研精神时时影响、感染和鼓励我的行为，并成为我今后学习的榜样。在此衷心感谢欧阳老师对我的悉心教导和帮助！

感谢知识工程教研室的所有老师、同学、师兄弟和师姐妹，在这个有凝聚力的研究集体中，我得到过许多人的帮助、关心和鼓励，在此表示深深的谢意。

感谢我的女友一直以来对我的支持，在学习上对我的鼓励、帮助，在生活中对我的关心和照顾，在我迷茫、困难的时候给了我勇气和前进的动力。

感谢我的父亲、母亲以及所有亲人，是你们的养育、期望和支持，使我顺利完成学业，在此向你们致以最诚挚的谢意。

感谢曾经给予我关心、支持的朋友和同学们。感谢所有曾经帮助过我的人，不论我们是否相识。

导师及作者简介

导师简介：

欧阳继红 女，1964 年生，博士，教授，博士生导师。主要研究方向为：知识工程与专家系统，时空表示与推理，数据挖掘，数据结构与算法。作为项目负责人承担科研项目 3 项，其中国家 863 项目 1 项，吉林省科技厅基金项目 1 项、完成吉林大学项目 1 项。作为骨干参加完成国家与省部级项目 16 项，其中国家重大项目 3 项。在国内外核心刊物上发表论文 40 余篇。作为第二参加人获省优秀教学成果一等奖 1 项、国家教委高校优秀教材二等奖 1 项，国家优秀精品课程 1 项。

作者简介：

龚资 男，1980 年出生，汉族，吉林省长春市人；硕士，吉林大学计算机科学与技术学院计算机应用技术专业，研究方向为：本体语义，本体描述语言及推理。在研究生就读期间，参与了国家自然科学基金重大项目和国家自然科学基金项目的研究。

Email: gongzi@sohu.com