

文章编号:1001-9081(2006)12-2800-03

对语义网查询语言的逻辑推理优化

陈家琪, 陶 媛

(上海理工大学 计算机工程学院, 上海 200093)

(taoyuan103@yahoo.com.cn)

摘 要:为了在 SeRQL RDF 查询语言的语境中对查询进行的转换和优化,提出了使用 Prolog 语言来处理推理策略原型的 RDF 图的方法,以及将 SeRQL 查询转化为优化 Prolog 目标的算法,实现了对语义网语言的结构化变换,并验证了 Prolog 模块的性能。

关键词:资源描述框架; SeRQL 查询; SWI-Prolog; 三元组

中图分类号: TP311.5 **文献标识码:** A

Using logic to optimise the semantic Web query language processing

CHEN Jia-qi, TAO Yuan

(College of Computer Engineering, University of Shanghai for Science and Technology, Shanghai 200093, China)

Abstract: In order to achieve query translation and optimization in the context of the SeRQL RDF query language, the method of using Prolog to manipulate RDF graphs was proposed, as well as the algorithm of translating SeRQL to Prolog goals. Besides, a structured mapping from language to semantics has been provided and the performance of the storage module for use in Prolog has been demonstrated.

Key words: Resource Description Framework (RDF); SeRQL query; SWI-Prolog; triple

0 引言

语义网的目标主要是建立一个基于本体工程和人工智能的简单三元组数据模型。资源描述框架(Resource Description Framework, RDF)是一种描述资源的语言,适用于任何领域,也是 W3C 组织推荐的描述万维网上元数据的标准。RDF 定义了一个简单模型,通过性质和值来描述资源以及资源与资源之间的关系。RDF 为定义和使用元数据提供了一个抽象的、概念化的框架,是处理元数据的基础^[1]。Sesame^[2]和它的查询语言 SeRQL^[4]是实现语义网 RDF 储存和查询系统的主要方法之一。

但是,在 Sesame 执行中遇到了几个问题:Sesame 把直接提供的三元组和从它们派生的三元组储存在一个数据库中,派生的三元组通过指定语义网语言(例如 RDFS)确定语域。这意味着将其转换到如 OWL-DL^[3]之类的语言就需要删除派生的三元组并对新语言重新计算推理闭包。Sesame 中的推理闭包在 RDFS 中是很小的,但是转换为 OWL 就会变得很大。Sesame 易于受查询路径表达式命令影响。虽然 Sesame 是通过 Java 编写的,但是使用 Prolog 语言处理新的推理策略原型的 RDF 图更加容易。

为了克服上面提到的问题,可通过 Prolog 模块实现包含多重推理引擎的服务器。用 SeRQL 语言阐述查询,然后通过以 Sesame HTTP 为基础的主从式协议转换查询其结果。通过 HTTP 上的 SeRQL 和查询优化器扩充储存和查询系统。

1 转换策略

1.1 阐述查询

通过执行 SWI-Prolog SeRQL,将一个 SeRQL 查询转换成

一个 Prolog 目标,其对象子图的边表现为对 RDF 的调用(主体,谓词,客体)。WHERE 语句通过在 SeRQL 运行模块中谓词的逻辑与、和来表示。编译器通过 DCG 解析器解析,然后处理 SeRQL 命名空间的描述并引进变量。

下面用 Prolog 语句表示 SeRQL 查询,并使用 SeRQL 查询中的变量名称,语句运行的语境如图 1 所示。这个继承模块支持谓词 RDF/3 的 Prolog 模块,可以产生并检查从现有的三元组派生的所有三元组。这意味着可以通过任何实例结构访问谓词,谓词绑定所有的变量,并通过继承法则的递归调用来产生所有的选择。如果 RDF/3 满足这些标准,则可以通过 Prolog 程序对 SeRQL 查询进行任意转换,并用它来处理查询。WHERE 语句的初始条件映射为继承模块中的谓词。

SeRQL 路径表达式在([...])中是随意的,这样的路径表达式通过 SWI-Prolog soft-cut 控制结构进行转换。它们绑定相匹配的变量,但是并不改变匹配图的核心。

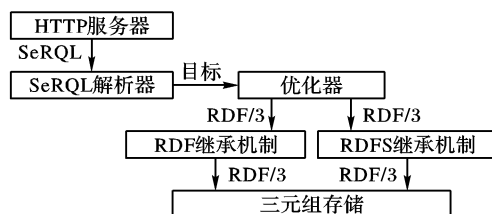


图 1 继承模块体系结构

1.2 排序问题

为了进一步研究排序问题,这里介绍一个在 WordNet^[5]上查询的例子。这个查询查找的是能被解释为至少两个文本类别的词,举例查询的结果如图 2 所示,在 WordNet 中,“sneeze”既可以是名词也可以是动词。

WordNet 是由 Synset 组成的,是一个按 wordForm 简要描

收稿日期:2006-06-30;修订日期:2006-09-01 基金项目:上海市教育委员会科研基金资助项目(04EB12)

作者简介:陈家琪(1957-),男,山东莱阳人,教授,主要研究方向:计算机网络通信与信息安全;陶媛(1981-),女,黑龙江人,硕士研究生,主要研究方向:计算机网络通信。

述的抽象实体。Synset 是 RDFS 的实例,也是 LexicalConcept 的一个子集。这里要查找一个属于两个不同 LexicalConcept 子集 Synset 的 wordForm。WordNet 的一些语法如表 1 所示。

表 1 WordNet 语法

语法成分	数量
WordNet metrics	
Distinct wordForms	123 497
Distinct synsets	99 642
wordForm triples	174 002
Subclasses of LexicalConcept	4

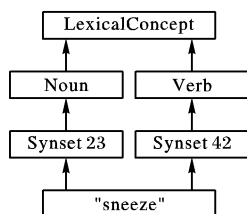


图 2 查询“sneeze”的结果

为了实例化优化需求和进行进一步研究,下面给出了这个查询的两种转化。直接转化和有选择可能的转化如示例 1 所示,它在 AMD 1600 + 处理器上运行需要 3.58s,有选择可能的转化则需要 8305s,慢了 2320 倍。有选择可能的转化同样可以直接转换成另外一个同样语义的 SeRQL 查询。

示例 1 WordNet 上查询的两种转换方式

```

s1(L): -
    rdf(S1, wns: wordForm, L), rdf(S2, wns: wordForm, L),
    rdf(S1, rdf: type, C1), rdf(S2, rdf: type, C2),
    rdf(C1, rdfs: subClassOf, wns: 'LexicalConcept'),
    rdf(C2, rdfs: subClassOf, wns: 'LexicalConcept'),
    C1 \== C2.

s2(L): -
    rdf(C1, rdfs: subClassOf, wns: 'LexicalConcept'),
    rdf(C2, rdfs: subClassOf, wns: 'LexicalConcept'),
    C1 \== C2,
    rdf(S1, rdf: type, C1), rdf(S2, rdf: type, C2),
    rdf(S1, wns: wordForm, L), rdf(S2, wns: wordForm, L).
  
```

在开始讨论优化之前,先解释一下为什么这些相同意义的程序执行时间会不同。假设有三个完全独立的文本 A, B, C, 对它们作逻辑乘,这里的独立是指它们之间没有变量是相同的。如果 $b()$ 表示一个文本解法的数量,则总的解法空间就是 $b(A) \times b(B) \times b(C)$,这样的话就是跟顺序无关的。如果不以解法空间而以访问状态作为评估尺度,公式就变成 $b(A) + b(A) \times b(B) + b(A) \times b(B) \times b(C)$,这个方法与 Prolog 逻辑推断的数量及对执行时间的正确预期是分不开的。它最先放置含有最少选择可能的文本,但是由于最终的组成才是最主要的,所以差别不大且不能解释示例 1 的两种转换。事实上第二种方法是在不考虑相关性的情况下对分歧因数进行排序的。

2 执行优化

2.1 估计复杂度

优化的第一步就是估计一个特定转化的复杂度。通过搜索树中访问节点的数量来估计执行时间,忽略执行不同 RDF/3 文本所需时间的一些不同。

对一个 RDF/3 访问的解决方法的数量进行估计:对每个 RDF/3 目标,零到多个参数中有一个值已知,且剩余的参数

是未绑定的,这样就可以很容易地估计其复杂度。如果没有已知的参数,则这个估计就是数据库中三元组的总数,并可以很容易通过数据库处理程序进行处理。如果所有的参数都是已知的,则设定它们的解答集合为 0.5。如遇其他情况则计算索引并返回其散列链的长度值。设定一个恰当的散列函数,用来对可提供的解答数目进行合理估计,并通过数据库进行处理。

估计谓词的分歧因数:通过执行文本绑定了变量,但是却不知道其确切值。仔细观察一下查询就会发现许多文本在查询时是知道谓词值的,这样就产生了两种情况:主体绑定一个未知值并且客体是未绑定的;或者情况相反。通过分别定义主体分歧因数 (Subject Branch Factor, SBF) 和客体分歧因数 (Object Branch Factor, OBF) 来对其进行处理,结果通过谓词进行储存,而且只有当三元组谓词的数量有相当大的改变时才进行重新计算。

连接词总的复杂度可以很容易的通过总搜索空间的大小进行表示。连接词每一步的分歧因数可以通过连接词的符号执行推理得到,用 Skolem 实例来替换文本中的变量。Skolem 实例使用 SWI-PROLOG 属性变量^[6] 执行。

2.2 优化连接词

通过一个对特定命令复杂度的良好估计,优化问题就可以简化成一个发生一测试问题。含有 N 个子句的连接词可以有 $N!$ 种不同的排序方法。由于有的现实实例的 N 接近 40, 直接转换是不可能的。由于不相关的子连接词可以独立确定,所以并不需要搜索所有的空间就可以对解答数目作出估计。

排序刚开始时,对于大部分的连接词所有的文本都是相关的。在执行一些文本之后,经常可以通过基本的变量将剩余的文本分成可被分别优化的独立组,算法 1 所示。

算法 1 对连接词进行排列,注意 select(), marked(*) 是非确定的。

```

order(conj)
{
    make_subgraphs(conj, subconjs);
    if (count(subconjs) > 1)
    {
        maplist(order, subconjs, ordered_subs);
        sort_by_complexity(ordered_subs, sorted);
        return join_subgraphs(sorted);
    }
    else
    {
        first = select(conj, rest); (*)
        skolem_bind(first);
        make_subgraphs(rest, subconjs);
        maplist(order, subconjs, ordered_subs);
        sort_by_complexity(ordered_subs, sorted);
        return first + join_subgraphs(sorted);
    }
}
  
```

将这个发生器与 1.3 部分的复杂性估计相结合,并用最好的方法来排序。由于这个排列算法只返回对独立子图进行重排序的结果,并通过在独立子图上进行分类来选取最好的排序,如果对复杂性估计好,则返回的这个排序就是最优化的。即最优化的排序和估计的排序的区别就在于估计函数可能有错误。

2.3 选择可能的路径表达式和控制结构

必须处理 Goal 和在 Prolog 控制结构中把 WHERE 语句作为个体进行编译得来的其他目标。如果这样的个体是连接词的话就递归调用排序算法。在输入连接词时一定要保证这个连接词的分类及估计复杂度所依据的变量是已知的。在控制结构中的连接词必须被排好并确定它们的复杂度,以便估计外部连接词的复杂度。

选择可能的路径表达式并没有改变查询的必要部分。它只是会产生更多的变量绑定。因此可以首先通过把连接词分为必要部分和可选部分来简化连接词的优化过程,然后对可选部分的必要部分进行优化。

2.4 处理独立路径表达式

在执行完一些文本后,由于查询的剩余部分被分成独立的子图,特殊排列的数量比可能排列的数量要少很多。这样就可以单独处理独立子图,且总的结果是所有部分结果的笛卡尔乘积。这个方法有以下几个优点:

1) 分别处理两个独立对象 A 和 B 的复杂度是 $b(A) + b(B)$ 而不是 $b(A) + b(A) \times b(B)$ 。

2) 如果任何一个独立对象没法解决,可以中断整个查询

并且判定它无解。

3) 子目标也可以采用类似的办法进行处理。

4) 结果集可以表示为部分结果的笛卡尔乘积,这样客户和服务器的通信就会减少很多。

5) 这样可以不需要运行算法 1 中的“sort by complexity”步骤。

可以在重排序后执行这个优化。它仅仅进行连接词的符号赋值和 Skolem 实例化,并将剩余部分划分成一系列子图。优化可以对产生的独立子图和在设计结构中的连接词进行递归调用。

3 实验结果

在两个域上对本文的优化进行评估,它们是上文提到过的 WordNet 和一个关于中国文化关系的 RDF 数据库。在一个双 AMD2600 + 的机器上运行 SuSE Linux 和 SWI-Prolog 5.5.15 来执行这个评估。

通过假设和现实的 SeRQL 查询来检验所进行的优化。在所有检验的例子中优化时间只占按最佳排序执行时间的一小部分,并且通常比解析查询所需的时间要短。

表 2 复杂查询的结果

序号	边	优化时间/ms	复杂度		提高的速度	总的时间/s	解答个数
			初始	最终			
1	37	9	1.4e16	1.4e10	1e6	2.47	79 778 496
2	29	9	2e13	1.3e5	1.7e8	0.49	3 826
3	28	9	1.4315	5.1e7	2.7e7	11.6	266 251 076

通过示例 1 的情形,优化器将两者中任意一种转换变成目标如算法 2 所示。s1/1 的代码可以看作是一种以求达到最佳性能的推测。这个测验证实了算法 2 的代码比 s1/1 的代码快 1.7 倍,并且可能是最快的。执行优化只需要 90ms,占最佳解决方案运行时间的 4.3%。

算法 2 优化后的 WordNet 查询。

q(L): -

```

rdf(S1, rdf: type, C1),
rdf(S1, wns: wordForm, L),
rdf(C2, rdfs: subClassOf, wns: 'LexicalConcept'),
rdf(S2, rdf: type, C2),
rdf(S2, wns: wordForm, L),
C1 \== C2,
rdf(C1, rdfs: subClassOf, wns: LexicalConcept))

```

第二组测试是通过在含有 97 431 个三元组的数据库上的三个查询进行的。Sesame 不能很好的执行所有的查询(只有一个可执行),随后的检测显示这些查询包含大量多重的子查询,其结果是一个巨大的笛卡尔乘积。可以将它们分成多重的查询,这样 Sesame 处理起来就容易多了。采用第 2.4 节的对独立路径表达式的解析,就不需要重写服务器。结果如表 2 所示。我们只能通过 Sesame 检验第二组查询,用时 132.72s 得到了 3 826 个解答。

4 结语

有不少对语义网采用逻辑处理的研究,如文献[7]采用 Denotational Semantics 来提供一个对语义语言的结构化变换。但这些方法大多注重的是正确性,而本文注重的则是结果和性能。

目前已经可以在语义网域中使用 Prolog 语言进行推理注

解^[8]。推理不是以正式的语义网语言为基础的,它使用 Ad hoc 定义的模型。通过执行 SeRQL 已经证明这种方案能完全有效地对 RDFS 进行处理,也证明了 SWI-Prolog 语言是支持属性变量的,它为图表、可扩展标示语言、RDF 三元组的储存提供了广阔的空间,而且 HTTP 可以建立多样的语义网应用程序,包括交互式的和网络服务器应用程序。

参考文献:

- [1] 宋伟,张铭. 语义网简明教程[M]. 北京: 高等教育出版社,2004.
- [2] BROEKSTRA J, KAMPMAN A, VAN HARMELEN F. Sesame: A generic architecture for storing and querying rdf and rdf schema[A]. Proceedings of the First International Semantic Web Conference, number 2342 in Lecture Notes in Computer Science[C]. Springer Verlag, 2002. 54 - 68.
- [3] DEAN M, SCHREIBER G, VAN HARMELEN F, et al. OWL Web ontology language reference[S]. Working draft, W3C, 2003.
- [4] WIELEMAKER J, SCHREIBER G, WIELINGA B. Prolog-based infrastructure for RDF: performance and scalability[A]. The Semantic Web - Proceedings ISWC'03[C]. Sanibel Island, Florida, Berlin, Germany, Springer-Verlag, 2003, LNCS 2870: 44 - 658.
- [5] MILLER GA. WordNet: A lexical database for english[J]. Communications of the ACM, 1995, 38(11): 39 - 41.
- [6] DEMOEN B. Dynamic attributes, their hProlog implementation, and a first evaluation[R]. Report CW 350, Department of Computer Science, KU Leuven, Leuven, Belgium, 2002.
- [7] PATEL K, GUPTA G. Semantic Processing of the Semantic Web[M]. Heidelberg: Springer Berlin, Lecture Notes in Computer Science, 2003. 80 - 95.
- [8] SCHREIBER G, DUBBELDAM B, WIELEMAKER J, et al. Ontology-based photo annotation[J]. IEEE Intelligent Systems, 2001, 16(3): 66 - 74.