온톨로지 추론 모델에 독립적인 SPARQL 추론 질의 처리를 위한 재작성 알고리즘

(A Rewriting Algorithm for Inferrable SPARQL Query Processing Independent of Ontology Inference Models)

정 동 원 † Yixin Jing †† 백 두 권 †††
(Dongwon Jeong) (Yixin Jing) (Doo-Kwon Baik)

요 약 이 논문에서는 SPARQL로 작성된 OWL-DL 온톨로지 질의에 대한 재작성 알고리즘을 제안한다. 현재 웹 온톨로지 저장소는 주어진 SPARQL 질의의 추론 결과를 얻기 위해 추론 온톨로지 모델을 생성하고 SPARQL 질의와 생성된 추론 온톨로지 모델과의 일치성을 비교한다. 추론 모델은 베이스 온톨로지 모델에 비해 보다 큰 공간을 필요로 하고 다른 추론 질의를 위해 재사용될 수 없기 때문에 앞서 언급한 접근 방법은 보다 방대한 크기의 SPARQL 질의 처리에 부적합하다. 이러한 문제점을 해결하기 위해 이 논문에서는 SPARQL 질의를 재작성하고 이를 기본 베이스 온톨로지 모델에 대해 질의 연산을 수행하여 결과를 획득할 수 있는 SPARQL 재작성 알고리즘을 제안한다. 이러한 목적을 이루기 위해, 먼저 OWL-DL 추론 규칙을 정의하고 이를 질의 그래프 패턴 재작성에 적용한다. 또한 추론 규칙들을 분류하고 이러한 규칙들이 질의 재작성에 미치는 영향에 대하여 기술한다. 제안 알고리즘의 장점을 보이기 위해, Jena 기반의 프로토타입 시스템을 구현한다. 비교 평가를 위해 테스트 질의를 이용하여 실험을 수행하고 제안 방법과 기존 접근 방법을 비교한다. 실험 결과에서, 제안 알고리즘이 완전성 및 정확성의 손실없이 메모리 공간 및 온톨로지 로딩 측면에서 향상된 성능을 보였다.

키워드 : 웹 온톨로지, SPARQL, 추론, OWL-DL, 질의 재작성, 그래프 패턴, 시맨틱 웹

Abstract This paper proposes a rewriting algorithm of OWL-DL ontology query in SPARQL. Currently, to obtain inference results of given SPARQL queries, Web ontology repositories construct inference ontology models and match the SPARQL queries with the models. However, an inference model requires much larger space than its original base model, and reusability of the model is not available for other inferrable SPARQL queries. Therefore, the aforementioned approach is not suitable for large scale SPARQL query processing. To resolve this issue, this paper proposes a novel SPARQL query rewriting algorithm that can obtain results by rewriting SPARQL queries and accomplishing query operations against the base ontology model. To achieve this goal, we first define OWL-DL inference rules and apply them on rewriting graph pattern in queries. The paper categorizes the inference rules and discusses on how these rules affect the query rewriting. To show the advantages of our proposal, a prototype system based on Jena is implemented. For comparative evaluation, we conduct an experiment with a set of test queries and compare of our proposal with the previous approach. The evaluation result showed the proposed algorithm supports an improved performance in efficiency of the inferrable SPARQL query processing without loss of completeness and soundness.

Key words : Web ontology, SPARQL, Inference, OWL-DL, Query Rewriting, Graph Pattern, Semantic Web

† 종신회원 : 국립군산대학교 수학정보통계학부 교수
djeong@kunsan.ac.kr
†† 학생회원 : 고려대학교 컴퓨터학과
jing@software.korea.ac.kr
††† 종신회원 : 고려대학교 컴퓨터학과 교수
baik@software.korea.ac.kr
논문접수 : 2008년 7월 31일
심사완료 : 2008년 10월 29일

정보과학회논문지: 데이타베이스 제35권 제6호(2008.12)

1. 서 론

시맨틱 웹에서 웹 온톨로지는 특정 도메인 내의 개념(Concepts)과 개념 간의 관계(Relationships)의 집합으로서, RDF(Resource Description Framework), RDF-S(RDF Schema) 및 OWL(Web Ontology Language)과 같은 온톨로지 서술 언어로 표현된다[1,2]. 웹 온톨로지는 개념과 관계성을 명세할 뿐 아니라 추론을 위해서도 이용된다. 추론은 웹 온톨로지의 필수적이고 기본적인 기능이자 장점이다. 이를 위해 베이스 온톨로지(Base Ontology, Original Ontology)에 추론 내용을 추가 및 확장함으로써 다른 온톨로지로 해석된다.

이러한 온톨로지 추론의 장점에 기인하여, Jena[3]나 Sesame[4]와 같은 대표적인 웹 온톨로지 구현 시스템들 모두 온톨로지를 해석하기 위해 RDF Semantics[5]나 OWL Entailment[2,5]와 호환되는 내장형 추론기를 제공한다. 해석이 완료되면 베이스 온톨로지 모델은 Tableau Expansion[6]이나 전향 추론 알고리즘[7]을 통해서 추론 모델로 저장된다.

그림 1은 앞서 언급한 기존 접근 방법의 처리 프로세스를 보여준다. 그림에서, 베이스 모델은 온톨로지 추론기에 의해 새로운 관계성(r1, r2, r3, r4)이 추가된 추론 온톨로지로 해석된다. r1은 http://www.Department0.University0.edu/FullProfessor0의 축약 표현인 FullProfessor0과 FullProfessor의 모든 상위 클래스들과의 묵시적 관계성을 선언한다. r2는 FullProfessor0과 http://www.University84.edu의 약식 표현인 Unversity84와의 관계성들인 degreeFrom 및 hasAlumnus를 선언적으로 정의한다. undergraduateDegreeFrom 프로퍼티가 degreeFrom의 하위 프로퍼티이고 degreeFrom과 hasAlumnus는 상호 역관계라는 사실로부터 University84를 유도하게 된다.

베이스 온톨로지를 추론 온톨로지로 해석하는 과정에서, 두 개의 추론 규칙이 적용된다. 첫 번째 추론 규칙은 다음과 같다.

"x가 y의 headOf고 y의 타입이 Department이면 x는 Chair 타입이다."

그림 1에서, FullProfessor와 http://www.Department0.University0.edu, 즉 Departement0 간에 headOf 관계성이 성립하기 때문에 추론기는 이 규칙을 이용하여 FullProfessor0과 Chair 간의 관계성을 생성한다 (그림 1의 r3).

두 번째 규칙은 University와 Lab 간의 관계성인 subOrgOf(sub-organization-of) 관계성(그림 1의 r4)을 생성하는 규칙으로서 이러한 관계성을 전이 프로퍼티(Transitive Property)라고 정의한다.

이러한 과정을 통해 생성된 추론 온톨로지는 사용자 질의에 대한 추론 결과를 반환한다. 예를 들어, University84의 alumnus를 검색하기 위한 질의가 주어지면 추론 온톨로지에 따라 FullProfessor0을 반환한다. 반면 베이스 온톨로지를 이용하여 질의 처리가 이루어질 경우 이러한 결과를 얻을 수 없다.

추론 온톨로지를 이용하여 묵시적인 정보에 대한 검색이 가능하지만 이러한 접근 방법은 몇 가지 문제점을 지닌다. 먼저, 온톨로지는 진화, 즉 지속적으로 확장되고 온톨로지 언어 표현력이 증가함에 따라 추론 온톨로지에 대한 종속성은 높은 로딩 및 유지 비용 문제를 발생시킨다[8,9]. 그림 1에서도 알 수 있듯이, 베이스 온톨로지를 추론 온톨로지로 해석하는 과정은 많은 시간을 요구하며, 또한 생성된 추론 온톨로지는 추가된 정보를 지니기 때문에 베이스 온톨로지에 비해 보다 많은 공간을 요구한다. 추론된 정보는 추론 규칙에 의해 결정된다. 이러한 추론 규칙은 OWL Entailment와 같은 표준화된 규칙일 수 있으며 또는 온톨로지 공학자에 의해 개발될 수 있다[10]. 보다 많은 온톨로지 규칙이 포함될수록 추론 온톨로지 크기가 증가한다. 무엇보다 온톨로지 크기가 증가할수록 온톨로지 질의 처리에 많은 시간이 소요된다. 이러한 문제점은 실험 환경에서는 용인될 수 있다. 그러나 실제 도메인에서는 매우 방대한 크기의 온톨로지를 이용하고 보다 추론 규칙이 복잡하므로 추론 온톨로지의 크기가 기하급수적으로 증가하기 때문에 결코 간과되어서는 안된다.

기존 접근 방법들의 두 번째 문제점은 해석되어 생성된 추론 모델을 재사용하지 못한다는 점이다. 즉, 하나의 추론 모델은 단지 하나의 추론 규칙 집합만을 지원한다. 그러나 실질적으로 하나의 온톨로지는 많은 사용자에 의해 사용되며 따라서 다양한 추론 요구(추론 질의)를 포함한다. 만일 온톨로지 추론기가 다른 추론 규칙들을 지원하고자 한다면 해당되는 다른 추론 온톨로지를 생성하여 별도로 저장해야 한다. 이러한 문제는 처리 성능에 있어서 추가적인 오버헤드를 지니며 아울러 많은 저장소 공간이 요구된다는 문제점을 야기한다.

이 논문에서는 질의 레벨에서 의미 추론을 구현하여 앞서 언급한 문제점을 해결할 수 있는 질의 재작성 알고리즘을 제안한다. 제안하는 알고리즘을 이용함으로써 추론 온톨로지 모델을 생성하지 않고 베이스 온톨로지 모델을 통해 질의 처리 결과를 얻을 수 있다. 이는 추론 온톨로지 생성에 따른 높은 로딩 시간, 높은 메모리 사용 및 온톨로지 규칙에 대한 의존성 등의 문제점을 해결할 수 있는 접근 방법이다.

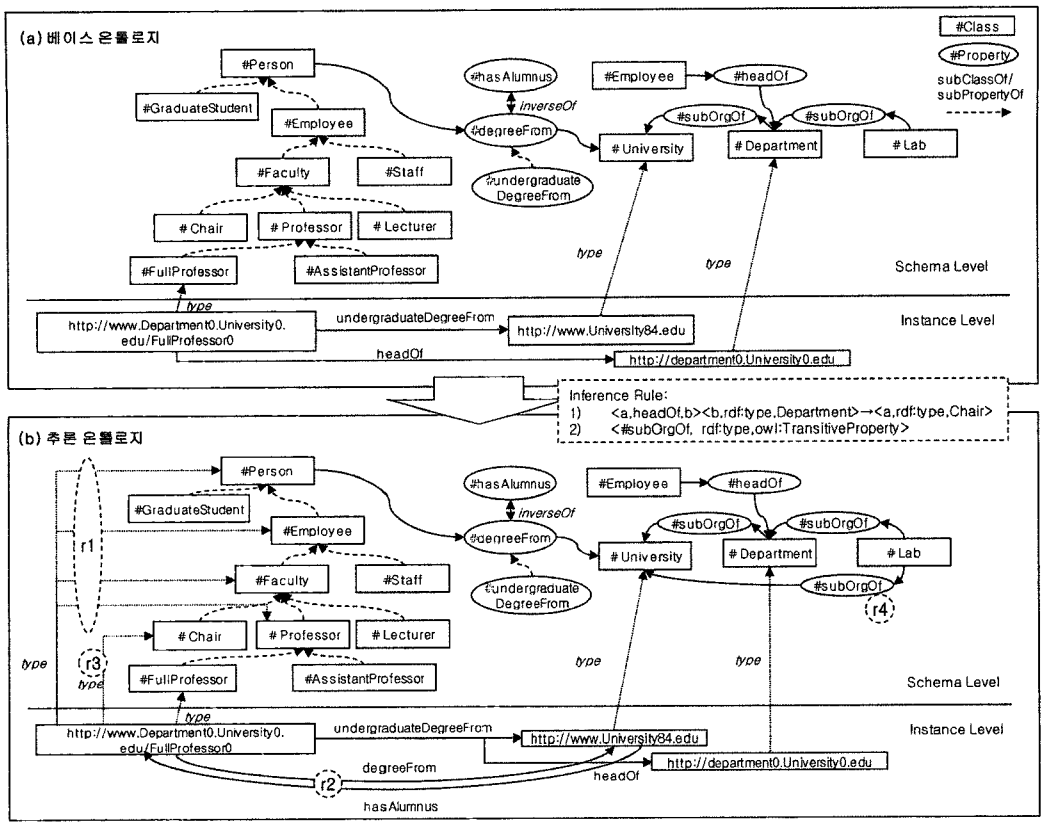

그림 1 베이스 온톨로지와 추론 온톨로지

이 논문의 구성은 다음과 같다. 제2장에서는 논문에 필요한 정의 및 주요 개념을 정의한다. 제3장에서는 이 논문에서 제안하는 접근 방법에 대하여 논의하고 제4장에서는 온톨로지 추론 규칙들에 대하여 기술한다. 제5장에서는 재작성 알고리즘에 대하여 상세하게 기술하고 제6장에서는 프로토타입 시스템 및 실험 결과를 보인다. 마지막으로 제7장에서는 결론 및 향후 연구에 대하여 서술한다.

2. 사전 정의

OWL로 작성된 온톨로지는 RDF 트리플(RDF Triples)로 변환되며, 따라서 RDF를 위한 질의 언어로서 개발된 SPARQL을 OWL 웹 온톨로지에 적용할 수 있다. 이는 웹 온톨로지가 그래프 모델이면서 트리플인 문장(Statements)의 집합이고, RDF나 OWL로 작성된 웹 온톨로지들 모두 트리플의 집합이라는 측면에서 SPARQL을 이용한 OWL에 대한 검색 연산이 가능하다. 이 절에서는 RDF와 SPARQL의 핵심 개념들을 대수적으로 정의한다.

I, L, B를 각각 IRI, RDF 리터럴, 공백 노드의 집합이라고 하자. 이 때 RDF 용어들의 집합 T는 (IULUB)이며 RDF 트리플(RDF Triple) RT는 다음과 같이 정의된다.

정의 1. RDF 트리플: RDF 트리플인 RT (s, p, o)는 (IUB) × I × (IUBUL)로 정의한다.

s, p, o은 각각 RDF 트리플의 subject, predicate, object를 나타낸다. 하나의 subject, s는 IRI이거나 공백 노드이며 p는 반드시 IRI이어야 한다. 하나의 object, o는 IRI, 공백 노드 혹은 리터럴 중 하나이다.

정의 2. RDF 그래프: RDF 그래프는 RDF 데이타 집합, 즉 RDF 트리플의 집합이다.

RDF 트리플은 온톨로지 모델을 해석하는 과정에서 생성된다. 만일 추론을 고려하지 않고 해석 연산이 완료되면 RDF 트리플은 온톨로지 내의 모든 정보(명시적인 정보)를 정확하게 표현한다. 그러나 만일 추론을 고려한 해석 연산이 실행되면 생성된 RDF 트리플은 온톨로지의 명시적인 정보는 물론 묵시적인 정보까지 포함한다. 이 논문에서는 이러한 두 가지 타입의 RDF 그래프를

각각 베이스 온톨로지 및 추론 온톨로지라 정의한다.

정의 3. 트리플 패턴(Triple Pattern): 하나의 트리플 패턴 t는 다음과 같이 정의된다.

$t \in (I \cup B \cup V) \times (I \cup V) \times (T \cup V)$

- V: SPARQL 질의 내에 있는 변수(Variables)의 집합

트리플 패턴에서, 즉 SPARQL 질의에서 변수는 subject, predicate, object의 모든 위치에 나타날 수 있다. 그림 1의 예제에서, 트리플 패턴의 한 예는 <?v, rdf:type, FullProfessor>로서 subject 부분에 변수가 기술된 경우이다.

정의 4. 그래프 패턴(Graph Pattern): 하나의 SPARQL 그래프 패턴 표현은 다음과 같이 순환적으로 정의된다.

(1) 하나의 트리플 패턴은 그래프 패턴이다.

(2) 만일 P_1과 P_2가 그래프 패턴이면, (P_1 AND P_2), (P_1 OPT P_2), (P_1 UNION P_2) 또한 그래프 패턴이다.

(3) 만일 P가 그래프 패턴이고 R이 SPARQL에 내장된 조건인 경우, (P FILTER R) 또한 그래프 패턴이다.

(4) 그래프 패턴은 SPARQL 질의 몸체 부분이다. 그래프 패턴은 트리플 패턴, 제약 조건 및 연산자(AND, OPT, UNION, FILTER)로 구성된다. 그림 1에서, <?v, rdf:type, FullProfessor > AND <?v, undergraduateDegreeFrom, University84 >는 그래프 패턴의 예이다.

정의 5. 솔루션 매핑(Solution Mapping, SM): 하나의 솔루션 매핑, μ는 부분 함수, $\mu : V \rightarrow T$로서 정의된다. μ(t)는 μ에 따라서 t내에 있는 변수를 대체함으로써 획득된 트리플이다. μ의 도메인, dom(μ)은 V의 부분 집합이다.

정의 6. 그래프 패턴 평가(Graph Pattern Evaluation) [18]: D를 통한 그래프 패턴 P의 평가는 [[P]]D에 의해 수행된다. 이 때, [[P]]D는 대수 연산자 join, left outer-join, union을 통해 연결된 μ의 집합이다. 특히, $[[t]]_D = \{\mu | dom(\mu) = var(t), \mu(t) \in D\}$이며 단, var(t)는 t내 변수들의 집합이다.

그림 1의 베이스 온톨로지를 이용하여 예를 들어 보자. 베이스 온톨로지의 해석 결과는 RDF 트리플의 집합, D이다. 베이스 온톨로지에 따라 D는 다음과 같은 두 개의 문장을 포함한다.

<FullProfessor0, rdf:type, FullProfessor>
<FullProfessor0, undergraduateDegreeFrom, University84>

이 때, 다음과 같은 질의가 주어졌다고 가정해 보자.

<?v, rdf:type, FullProfessor >
AND <?v, undergraduateDegreeFrom, University84>

주어진 질의는 D에 일치되며, 결론적으로 그 결과로서 FullProfessor0이 반환된다. 위 과정을 정형적으로 표현하면 다음과 같다.

[[<?v, rdf:type, FullProfessor >AND <?v, undergraduateDegreeFrom, University84 >]]$_D$
= [[<?v, rdf:type, FullProfessor >]]$_D$
AND [[<?v, undergraduateDegreeFrom, University84 >]]$_D$
= {FullProfessor0}AND{FullProfessor0}
= {FullProfessor0}

3. 그래프 패턴 재작성 접근 방법

제안하는 접근 방법의 개념을 보이기 위해 University 84의 alumnus를 검색하는 SPARQL 질의를 다시 고려해 보자. degreeFrom과 hasAlumnus 간의 역 관계성과 degreeFrom과 undergraduateDegreeFrom 간의 상하 관계성에 따라서 University84 degree와 undergraduate라는 degree를 획득한 사람을 검색하는 새로운 질의를 생성할 수 있다. 이렇게 생성된 질의는 추론 온톨로지가 아닌 베이스 온톨로지에 대해 질의 처리가 이루어진다. 이러한 방법으로 추론 결과인 FullProfessor0이 검색된다.

이 논문에서 제안하는 이러한 접근 방법은 추론 온톨로지를 생성하지 않으며 따라서 많은 처리 비용을 절감할 수 있다. 또한 베이스 온톨로지를 모든 온톨로지 사용자들이 재사용할 수 있다. 위와 같이 새로운 SPARQL 질의를 다시 작성하는 처리 과정을 SPARQL 재작성이라고 정의한다.

질의 재작성 개념은 불일치하는 데이타베이스 간 통합을 위해 데이타베이스 분야[11]에서 이미 널리 적용되었으며 최근 시맨틱 웹 분야[12]에서도 활용되고 있다. 질의 재작성은 뷰라고 정의되는 일관성 있는 데이타 스키마를 통해 불일치하는 데이타들을 액세스하기 위한 수단이다[13]. 그러나 이 논문에서의 질의 재작성 개념은 데이타 통합에서의 개념과는 다르게 정의되어야 한다. 가장 중요한 차이점은 논문에서 정의한 질의 재작성은 다중 데이타 셋으로부터 데이타를 검색하기 위한 목적이 아닌 온톨로지의 묵시적 결과까지 획득하기 위해 질의를 확장한다는 점이다. 결과적으로, 이 논문에서의 질의 재작성은 일반적인 질의 재작성과는 다르다.

질의 재작성은 질의 언어와 밀접한 연관성을 지닌다. 이 논문에서는 다양한 온톨로지 질의 언어들 중에서 [14-17], OWL[2] 온톨로지를 처리하는 웹 온톨로지 시

스템에서 가장 널리 사용되고 있는 W3C의 권고안인 SPARQL[14]을 이용한다. 이 논문의 제안 방법에서, 주어진 SPARQL 질의는 추론 규칙에 따라 재작성된다. 즉 논문에서 제안하는 재작성 알고리즘은 주어진 SPARQL을 정확하게 확장할 수 있음을 의미한다.

앞서 언급하였듯이, 기존 접근 방법과 달리, 이 논문에서는 명시적인 결과는 물론 묵시적인 결과를 질의 처리 엔진이 반환할 수 있도록 그래프 패턴을 확장하는 접근 방법을 이용한다. 이를 위해 SPARQL의 연산자인 UNION을 이용하며 UNION을 이용하는 이유는 다음과 같다.

UNION은 하위 그래프 패턴을 통합하기 위한 SPARQL의 연산자로서, 질의의 최종 결과(Query Solution)는 각 하위 그래프 패턴들에 대한 합집합이다. 임의의 트리플 패턴이 다른 그래프 패턴과 UNION 연산을 수행해도 검색 결과에 영향을 주지 않는다. 이를 정형적으로 증명해 보이면 다음과 같다.

정리 1. 그래프 패턴 P가 주어졌을 때, P와 다른 그래프 패턴과의 UNION 연산 결과로 생성된 트리플 패턴에 의해 유도된 그래프 패턴을 P_r이라고 할 때, $[[P]]_D \subseteq [[P_r]]_D$이다.

증명. [18]에서의 정의를 적용하면 다음과 같이 순환적으로 증명할 수 있다.

(1) If P = t, $[[t \text{ UNION } P']]_D = [[t]]_D \cup [[P']]_D \supseteq [[t]]_D$;

(2) If $P = (P_1 \text{ AND } P_2)$,

$[[(P_1 \text{ UNION } P'_1) \text{AND} (P_2 \text{ UNION } P'_2)]]_D$

$= [[(P_1 \text{ AND } P_2) \text{UNION} (P'_1 \text{ AND } P_2) \text{UNION} (P_1 \text{ AND } P'_2) \text{UNION} (P'_1 \text{ AND } P'_2)]]_D$

$= [[P_1 \text{ AND } P_2]]_D \cup \ldots$

$\supseteq [[P_1 \text{ AND } P_2]]_D$;

(3) If $P = (P_1 \text{ OPT } P_2)$,

$[[(P1 \text{ UNION } P'_1) \text{OPT} (P_2 \text{ UNION } P'_2)]]_D$

$= [[(P_1 \text{ OPT } P_2) \text{UNION} (P'_1 \text{ OPT } P_2) \text{UNION} (P_1 \text{ OPT } P'_2) \text{UNION} (P'_1 \text{ OPT } P'_2)]]_D$

$= [[P_1 \text{ OPT } P_2]]_D \cup \ldots$

$\supseteq [[P_1 \text{ OPT } P_2]]_D$;

(4) If $P = (P_1 \text{ UNION } P_2)$,

$[[(P_1 \text{ UNION } P'_1) \text{UNION} (P_2 \text{ UNION } P'_2)]]_D$

$= [[(P_1 \text{ UNION } P_2) \text{UNION} (P'_1 \text{ UNION } P'_2)]]_D$

$= [[P_1 \text{ UNION } P_2]]D \cup [[P'_1 \text{ UNION } P'_2]]_D$

$\supseteq [[P_1 \text{ UNION } P_2]]_D$.

FILTER 연산자 뒤에 오는 그래프 패턴들과 같은 내장된 조건들은 질의 결과를 필터링하기 때문에 재작성과 관련성이 없다. 내장된 조건에 대한 변경은 질의 결과에 영향을 준다.

이미 앞서 언급했듯이, 전체적인 재작성 프로세스는 다음과 같다.

(1) OWL-DL과 사용자 정의 추론 규칙이 저장소에 저장된다.
(2) 입력된 질의의 트리플 패턴들을 분석한다. 분석된 패턴에 따라 재작성 연산을 수행하기 위해 UNION을 어떻게 적용할지를 결정한다.
(3) 재작성 된 질의를 베이스 온톨로지 모델에 대해 실행하여 결과를 생성한다.

재작성 프로세스의 각 단계에 대해서는 4장 및 5장을 통해 보다 상세하게 기술한다.

4. 온톨로지 추론 규칙

이 장에서는 그래프 패턴, 즉 SPARQL 질의 재작성을 위한 온톨로지 추론 규칙을 정의한다. 온톨로지 추론 규칙들은 그래프 패턴을 재작성하기 위해 정의된다. 재작성의 결과는 주어진 그래프 패턴의 확장된 그래프 패턴이다. 추론 규칙들은 OWL-DL Entailment를 통해서 정의되거나 OWL Entailment에 의해 지원되지 않는 추론을 위해 온톨로지 사용자에 의해 정의될 수 있다. 추론 규칙의 의미에 따라 이러한 규칙들을 각각 CD(Conclusion), H(Hierarchy), P(Pair), TC(Transitive Chain)로 분류할 수 있다.

CD는 조건부와 결론부로 구성된다. 조건은 UNION으로 연결된 요소 조건(Element Conditions)의 집합으로 각각이 하나의 RDF 그래프이다. 이 그래프들의 의미는 베이스 모델과 일치한다. 결론부는 RDF 트리플로서 논리적으로 요소 조건으로부터 연역 추론된다. 추가적으로, 결론부에 있는 subject나 object는 각 요소 조건에도 나타난다. 정형화 된 정의(Formal Definition)는 다음과 같다.

정의 7. Conclusion C_D 정의: 베이스 온톨로지 모델 D가 주어졌을 때, C_D는 2-튜플로서 다음과 같이 정의된다.

C_D = (Con, r:(v_1, v_2, v_3)), where
- Con (Conditions) = UNION c_i
- c_i는 요소 조건이며 c_i = AND (l:(v_{ij1}, v_{ij2}, v_{ij3}))
- l(left)과 r(right)은 각각 규칙의 조건부와 결론부를 의미
- 각 c_i에 대해, $c_i \rightarrow$ r:(v_1, v_2, v_3)이고 {r:v_1, r:v_3} $\cap$ {l:v_{ij1}, l:v_{ij3}} $\neq \phi$

이 논문에서 가독성을 위해 결론부를 UNION $c_i \rightarrow r : (v_1, v_2, v_3)$으로 표현하며 예제는 다음과 같다.

예제 1. (a, headOf, b)(b, rdf:type, Department) → (a, rdf:type, Chair)
예제 1의 규칙은 사용자에 의해 정의된 추론 규칙(Customized Inference Rule)의 예로서 하나의 요소 결론만을 지닌다.

예제 2. (a, rdf:type, b)(b, rdfs:subClassOf, c) → (a, rdf:type, c)
예제 2는 RDF Entailment 규칙 중 하나로서 a의 rdf:type이 b이고 b가 c의 하위 클래스, 즉 rdf:subClassOf 관계에 있을 경우, a의 rdf:type이 c라는 결론을 유추할 수 있다.
H(Hierarchy)는 요소 그룹 내에 있는 명시적인 상하 관계성(Super-Sub Relationship)을 통해 형성된 구조를 표현한다. 이러한 관계는 클래스들 간 혹은 프로퍼티 간에 존재한다. H는 다음과 같이 정의된다.

정의 8. Hierarchy H 정의: $C=\{c_1, c_2, ..., c_n\}$를 가정할 때, H는 다음과 같이 정의할 수 있다.
$H = \{h=\{c_i[start, end]\}, 1 \le i \le n$이고 start, end ∈ Integer 혹은 $(c_i, subOf, c_j)$이면 $c_i.start > c_j.start$ & $c_i.end < c_j.end\}$

H는 h의 집합으로 구성되며 하나의 h는 ci의 집합으로 구성된다. 트리 내의 해당 위치에 따라 [start, end] 값이 c_i에 할당된다[19]. 특히 c_i가 rdf:Class 타입인 경우에는 이러한 H를 HC로 표기하며, 반면 c_i의 타입이 rdf:Property인 경우에는 HP로 표현한다. 위 정의에서 subOf는 subClassOf와 같이 상하 관계를 표현하는 프로퍼티의 집합을 의미한다.
따라서 만일 $c_i \in h$이면, 다음 함수들은 각각 c_i의 상위와 하위 노드를 반환한다.

$findSuper(c_i) = \{c' | c' \in h, c'.start < c_i.start,$ and $c'.end > c_i.end\}$
$findSub(c_i) = \{c' | c' \in h, c'.start > c_i.start,$ and $c'.end < c_i.end\}$

예제 3. 그림 2는 그림 1에 표현되어 있는 클래스 계층을 보여준다.

P(Pair)는 2-튜플로서 각 튜플은 키 대칭 프로퍼티(Key Symmetric Property)와 해당 키 프로퍼티에 영향을 받는 클래스, 인스턴스(Individuals) 및 프로퍼티들의 쌍을 저장하는 집합, S로 구성된다.

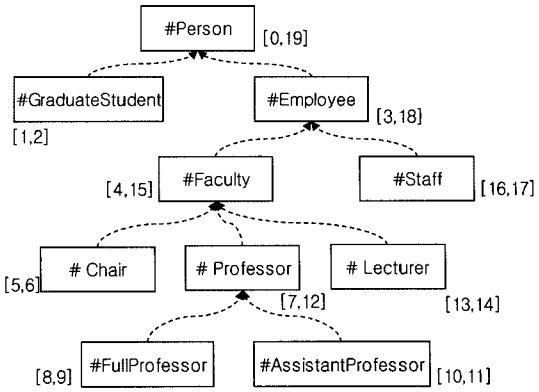

그림 2 클래스 계층 예제

정의 9. P는 다음과 같이 정의된다.
P_{key} = (key, S),
- key ∈ {rdfs:equivalentClass, rdfs:equivalentProperty, owl:inverseOf, owl:sameAs, owl:differentFrom, owl:disjointWith, owl:complementOf}
- $S = \{(e_1, e_2) | (e_1, key, e_2)\}$

다음은 P의 예를 보여준다. 주어진 예제에서 hasAlumnus와 degreeFrom이라는 두 프로퍼티는 상호 owl:inverseOf 관계성을 지님을 의미한다.

$P_{inverseOf}$ = (owl:inverseOf, {(hasAlumnus, degreeFrom)})

P는 온톨로지 모델에 명시적으로 정의되어 있는 문장들을 정의한다. 일부 대칭 관계성(Symmetric Relationships)은 Entailment 규칙에서 유도된다. 이러한 유도 문장은 P가 아닌 CD, 즉 Conclusion에 의해서 관리된다.
마지막으로, 중요한 요소는 전이 프로퍼티(Transitive Properties)이다. 전이 프로퍼티들은 Transitive Chain (TC)에 저장되며 다음과 같이 정의된다.

정의 9. Transitive Chain, TC = {p|(p, rdf:type, owl:TransitiveProperty)}

5. 트리플 패턴 재작성

하나의 트리플 패턴은 서로 다른 형태를 지니며 온톨로지 추론 규칙을 재작성 연산 수행을 위해 어떻게 적용할지를 결정한다. 이 장에서는 다양한 트리플 패턴 유형에 대한 재작성 방법에 대하여 기술한다.
그림 3은 전체적인 프로세스를 보여준다. 먼저 트리플 패턴 유형을 식별한다. 패턴 유형에 따라 다양한 트리플 패턴 처리 방법이 요구된다. 변수가 subject 또는 object

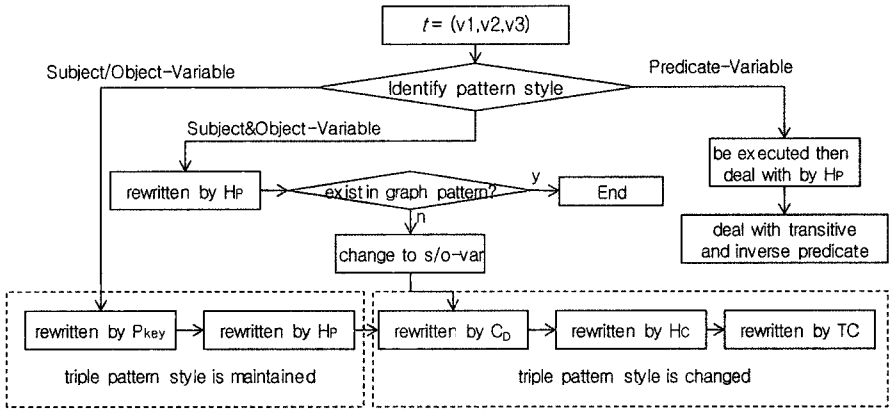

그림 3 재작성을 위한 전체적인 알고리즘

부분 중 한 곳에 나타나는 형태인 subject/object-variable 트리플 패턴을 고려해 보자. 먼저 트리플 패턴 유형을 변경하지 않는 Pair와 H_P에 의한 재작성 연산이 수행된다.. 그 다음으로 C_D, H_C, TC를 포함하는 패턴이 패턴 유형을 변경하는 프로세스들을 통해 재작성된다.

subject와 object 두 위치에 변수를 포함하는 Subject&Object-Variable 트리플 패턴의 경우, H_P(프로퍼티 계층)에 의해 재작성된다. 다음 연산은 다음과 같이 두 가지 상황을 고려해야 한다.

(1) 만일 트리플 내의 변수가 다른 트리플에 존재하지 않는 경우: 이 경우에, 트리플은 Subject&Object-Variable 트리플 패턴으로 변경된 후에 추론 규칙에 의해 재작성된다.

(2) 트리플 내 변수가 다른 트리플에 존재하는 경우: 이 경우에는 재작성 프로세스가 종료된다.

Predicate-Variable 트리플 패턴의 경우, 우선적으로 실행이 이루어지고 H_P에 따라 묵시적인 결과를 얻게 된다. 이와 같은 재작성 과정은 트리플에 포함된 변수의 개수와 그 위치에 따라 분류된다. 또한 주어진 트리플 원형의 보존 여부에 따라 재작성에 적용되는 순서가 결정된다. 다시 말해, 변수가 하나인 경우, 그 위치가 subject 혹은 object 부분에 나타나는 경우에는 Pair에 의한 재작성 연산이 수행되고 predicate 위치에 나타나는 경우에는 H_P에 의한 재작성 연산이 수행된다. 이 때, 이 두 연산은 상호 독립적이므로 적용 순서에 영향을 주지 않는다. 반면, 변수가 subject와 object 위치에 동시에 나타나는 경우에 대해서는 C_D, H_C, TC에 의한 재작성이 수행되며, 이 결과는 주어진 트리플 패턴의 원형을 변화시킨다. 만일 C_D, H_C, TC에 의한 재작성이 먼저 수행된다 해도 질의 결과에 대한 완전성에는 변화가 없다. 그러나 Pair와 H_P에 입력으로 주어지는 트리플 후보군이 증가하게 되어 성능적인 차이를 가져올 수 있다. 앞서 소개한 전체적인 트리플 패턴 재작성 과정은 이후 절들을 통해서 상세하게 기술한다.

5.1 Subject/Object-Variable 트리플 패턴

Subject/Object-Variable 트리플 패턴은 하나의 변수만을 지니는 형태이다. 이러한 트리플 패턴은 다양한 추론 규칙과 일치할 수 있으며, 새로운 트리플 패턴 유형을 생성하지 않기 때문에 이에 대한 재작성은 Pair와 H_P를 통해 우선적으로 수행된다. subject나 object 위치에 하나의 변수만을 포함하는 트리플 패턴 $t=(v_1,v_2,v_3)$가 주어졌을 때, v_2는 $P_{inverseOf}$와의 비교 연산이 이루어진다. 만일 v_2가 역 관계 프로퍼티 (Inverse Property)인 v'_2를 지니면 새로운 트리플 패턴 $t'=(v_3, v'_2, v_1)$을 t와 UNION 연산을 수행하게 된다(t UNION t′). 이후에, v_2와 v'_2간에 H_P와 PequivalentProperty에 의해 평가 연산이 이루어진다. v_2와 v'_2가 H_P내에 존재하면 v_2와 v'_2를 각각의 하위 프로퍼티로 대체하는 트리플 패턴들을 UNION 연산을 통해 확장한다.

다음은 앞서 언급한 프로세스를 표현한 알고리즘이다.

Algorithm-1. 역, 하위, 동치 프로퍼티 관계성 평가를 통한 재작성 알고리즘

```
procedure PropertyRewrite
    graph ← (v1, v2, v3);
    if (v2, v'2) ∈ P_inverseOf then
        graph ← (v1, v2, v3) UNION (v3, v'2, v1);
    end if
    ... /** 나머지 P_key에 대해서 위와 같은 동일한 연산이 수행됨 */
    if v2 ∈ HP then
        Set s ← findSub(v2);
        for ∀v ∈ s do
            graph ← graph UNION (v1, v, v3);
        end for
    end if
    ... /** v'2에 대해서 위와 같은 과정을 반복 수행 */
end procedure
```

위 Algorithm-1에 의한 연산이 완료되면, 베이스 트리플 패턴은 하나의 새로운 그래프 패턴으로 강화(Enrichment)된다. 각각 새롭게 생성된 트리플 패턴들

은 단순히 subject 혹은 object 위치에 변수가 하나만 존재하는 Subject/Object-Variable 패턴으로 subject 및 object 위치에 대한 추론 규칙을 적용하지 않은 상태이다.

다음 단계는 각 트리플 패턴에 클래스와 관련된 규칙을 적용하기 위한 과정이다. 이를 위해 먼저 Conclusion, C_D에 대하여 다룬다. 재작성 연산을 수행하기 전에, 트리플이 C_D 내의 규칙과 일치하는지를 평가해야 한다. 예제로서 Subject-Variable 트리플 패턴, 즉 subject 위치에 변수를 포함하는 트리플 패턴인 $t=(?v_1,v_2,v_3)$을 고려하여 평가 알고리즘을 정의하면 다음과 같다.

정의 10. Subject-Variable 트리플 패턴과 C_D 일치 평가 조건 정의: 임의의 Subject-Variable 트리플 패턴 $t = (?v_1, v_2, v_3)$가 주어졌을 때, 다음 조건을 만족하면 t가 C_D와 일치한다.

iff $r{:}v_1 \in \{r{:}v_1, r{:}v_3\} \cap \{l{:}v_{ij1}, l{:}v_{ij3}\}$ & $r{:}v_2 = t.v_2$

만일 임의의 t가 CD와 일치하면, 규칙 중 조건부인 그래프 패턴이 t와 UNION 연산이 이루어진다. 조건부의 subject와 object는 변수로 변경된다. 특히 t의 변수에 해당하는 subject들은 t의 동일한 변수로 변경된다. object 부분이 변수인 $t = (v_1, v_2, ?v_3)$인 경우도 앞선 과정과 동일하다. Algorithm-2는 Subject-Variable 트리플 패턴에 대한 C_D를 적용하는 알고리즘을 보여준다.

Algorithm-2. Conclusion, C_D 규칙 적용을 통한 그래프 패턴 재작성 알고리즘

```
procedure ConclusionRule
    if t = (?v1, v2, v3) matches CD then
        graph ← l:(vij1, vij2, vij3);
        change the l:vij1 that corresponds to r:v1 to ?v1;
        for other l:vij1, l:vij3 do
            change them to ?vij1, ?vij3;
        end for
        graph ← t UNION graph;
    end if
end procedure
```

다음은 Algorithm-2를 적용하여 Subject-Variable 트리플 패턴을 확장하여 재작성하는 예를 보여준다.

예제 5. 먼저 다음과 같은 트리프 패턴, t와 CD를 가정해 보자.

$t = (?x, rdf{:}type, Professor)$

$C_D = (a, rdf{:}type, b)(b, rdfs{:}subClassOf, c)$

$\rightarrow (a, rdf{:}type, c)$

다음과 같은 조건이 주어졌을 때, 주어진 t는 C_D와 일치한다.

$r{:}v_1 = a$

$r{:}v_3 = c$

$a \in \{r{:}v_1, r{:}v_3\} \cap \{l{:}v_{ij1}, l{:}v_{ij3}\} = \{a, c\}$

$r{:}v_2 = t.v_2 = rdf{:}type$

따라서 주어진 t는 다음과 같이 확장되어 재작성된다.

(?x, rdf:type, Professor)⇒ (?x, rdf:type, Professor) UNION ((?x, rdf:type, ?b)AND(?b, rdfs:subClassOf, Professor))

C_D에 기반한 재작성이 완료되면, 클래스 계층을 고려하여 트리플에 대한 재작성 연산을 수행한다. 예를 들어, rdfs:subClassOf가 트리플 패턴에 나타나는 다음과 같은 질의의 경우, 클래스 계층을 고려하여 재작성이 이루어져야 한다.

$(?v_1, rdfs{:}subClassOf, v_3)$ 또는 $(v_1, rdfs{:}subClassOf, ?v_3)$

이러한 형태의 트리플에 대한 재작성은 H_C 내의 규칙을 통해 이루어진다. 위의 트리플 예제 $(?v_1, rdfs{:}subClassOf, v_3)$과 $(v_1, rdfs{:}subClassOf, ?v_3)$의 경우, 각각 $findSub(v_3)$ 및 $findSuper(v_1)$ 함수를 통해 간단하게 결과를 얻을 수 있다. 그 결과에 따라서 그래프 패턴은 UNION 연산자를 이용하여 재작성된다.

다음은 위 과정을 명확하게 보이기 위한 예제이다.

예제 6. 예제 5의 예제를 고려해 보자. 만일 findSub(Professor)의 결과가 {FullProfessor, AssistantProfessor}이면 주어진 트리플 패턴(?b, rdfs:subClassOf, Professor)은 다음과 같이 재작성된다.

((?x, rdf:type, ?b)AND(?b, rdfs:subClassOf, Professor)) ⇒ (?x, rdf:type, FullProfessor) UNION (?x, rdf:type, AssistantProfessor)

Subject/Object-Variable 트리플 패턴의 재작성을 위한 마지막 단계는 전이 프로퍼티 관계를 나타내는 owl:TransitiveProperty를 고려하는 단계이다. Subject-Variable 트리플 패턴 $t=(?v_1, v_2, v_3)$을 고려해보자. 이때, t는 조건 $t.v_2 \in TC$를 만족하여 TC와 일치한다. 이러한 경우, $t.v_2$의 subject로 연결된 모든 노드를 탐색해야 한다. 사전 정의된 클래스 계층과 달리, 클래스 인스턴스 계층(Individual Hierarchy)은 인스턴스 간 전이 프로퍼티에 따라서 사전에 구축되지 않는다. 따라서 $t.v_3$의 상위 노드를 검색하기 위해서는 실시간 연산이 이루어져야 한다. 완전성과 효율성, 즉 모든 인스턴스의 검색과 보다 나은 검색 성능을 제공하기 위하여 2 단계 순회를 통해 TC와 일치하는 트리플 패턴을 확장한다. 트리플 패턴이 확장된 이후에, 만일 보다 하위 결과가 존재하게 되면, 즉 하위 노드가 검색이 되면 이러한 연산이 반복적으로 수행된다.

다음은 앞서 기술한 과정을 예제를 통해 보여준다.

예제 7. 트리플 패턴 t = (?v, subOrgOf, University)이면 이는 다음과 같이 확장된다.

t = (?v, subOrgOf, University)

⇒ t UNION ((?v′, subOrgOf, ?a) (?a, subOrgOf, University)) = t'

만일 ?v'의 값들 중에서 하위 노드를 지니는 클래스가 존재한다면 위와 같은 과정을 반복적으로 수행하게 된다. 예를 들어, ?v'의 값이 Lab이고 Lab의 하위 노드가 존재한다면 (?v, subOrgOf, Lab) 패턴에 대해 위와 같은 동일한 연산이 반복적으로 수행된다. 이러한 반복적인 연산은 ?v'의 값들이 단말 노드일 때까지, 즉 하위 노드를 지니지 않는 노드들의 집합으로만 구성될 때까지 수행된다. Lab이 하위 노드를 지니는 경우, 확장된 질의는 다음과 같다.

t"=t' UNION ((?v", subO, ?b)(?b subO, Lab))

5.2 Subject&Object-Variable 트리플 패턴

Subject&Object-Variable 트리플 패턴은 subject와 object 부분에 변수를 지니는 형태이다. 트리플 패턴 t= ($?v_1$, v_2, $?v_3$)이 주어지고 만일 $t.v_2 \in HP$이면 $t.v_2$의 하위 프로퍼티를 탐색하는 연산(findSub($t.v_2$))을 수행함으로써 재작성 연산을 시작한다.

반환된 프로퍼티를 υ라 할 때, ($?v_1$, υ, $?v_3$)과 같은 패턴을 생성하게 된다. 따라서 주어진 t는 다음과 같이 확장된다.

($?v_1$, v_2, $?v_3$) ⇒ ($?v_1$, v_2, $?v_3$)UNION($?v_1$, υ, $?v_3$)

그러나 재작성 된 패턴들은 여전히 Subject&Object-Variable 트리플이다. 이와 같이 확장된 트리플 패턴을 대상으로, 이 패턴들이 지니고 있는 변수들, 즉 $?v_1$ 혹은 $?v_3$이 다른 트리플 패턴 내에 포함되는지를 평가하게 된다. 만일 이러한 경우를 지니지 않는다면 t는 $?v_1$과 $?v_3$의 결과 집합인 μ($?v_1$)과 μ($?v_3$)을 얻기 위해 평가된다. 그리고 나서 재작성 과정은 5.1절에 기술한 과정을 통해 (μ($?v_1$), v_2, $?v_3$)과 ($?v_1$, v_2, μ($?v_3$))에 대한 재작성 연산이 강화된다. 즉, 5.1절에서 기술한 연산과 동일한 과정을 (μ($?v_1$), v_2, $?v_3$)과 ($?v_1$, v_2, μ($?v_3$)) 각각에 적용한다.

예제 8. 앞선 연산 과정을 보이기 위해 다음 질의를 고려한다.

($?v_1$, degreeFrom, $?v_2$)

이 때, undergraduateDegreeFrom 프로퍼티를 degreeFrom 프로퍼티의 하위 프로퍼티라고 하면, 질의는 다음과 같이 재작성된다.

($?v_1$, degreeFrom, $?v_2$) UNION ($?v_1$, undergraduateDegreeFrom, $?v_2$)

베이스 온톨로지에 대한 질의 실행 결과가 다음과 같다고 가정하자.

R[($?v_1$, degreeFrom, $?v_2$)] = NULL

R[($?v_1$, undergraduateDegreeFrom, $?v_2$)]

= ($?v_1$=FullProfessor0) & ($?v_2$= University84)

마지막으로, 결과가 반영된 두 개의 트리플 패턴이 다음과 같이 생성되며, 이를 5.1절에서 기술한 재작성 알고리즘을 적용함으로써 최종적으로 재작성 된 트리플 패턴을 얻을 수 있다.

(FullProfessor0, undergraduateDegreeFrom, $?v_2$)

($?v_1$, undergraduateDegreeFrom, University84)

5.3 Predicate-Variable 트리플 패턴

Predicate-Variable 트리플 패턴은 predicate 위치에 변수를 포함하는 패턴을 의미한다. 트리플 패턴 t = (v_1, $?v_2$, v_3)이 주어졌을 때, $?v_2$에 대한 완전한 결과를 위해 두 가지 형태의 처리가 이루어져야 한다. 먼저 t와 일치하는 트리플들을 검색하여 $?v_2$의 결과 집합을 획득한다. 획득된 결과 집합인 μ($?v_2$)의 상위 프로퍼티를 검색하기 위해 findSuper(μ($?v_2$))가 실행된다. 예를 들어, 그림 1에서, undergraduateFrom의 상위 프로퍼티인 degreeFrom을 검색한다. 이 때, degreeFrom과 hasAlumnus 간 inverseOf 관계가 성립하고 이러한 관계를 통해 v_1과 v_3간에 hasAlumnus 관계에 있는 트리플 패턴에 대한 질의 처리가 가능하게 된다.

위 연산을 통해 v_1과 v_3 간 직접적으로 연결된 프로퍼티와 프로퍼티들의 상위 프로퍼티에 대한 검색이 완료되면, t의 v_1과 v_3을 평가하게 된다. v_1이 인스턴스이고 v_3이 클래스인 경우, v_1과 v_3 간 묵시적인 rdf:type을 추출하기 위해 다음 질의를 실행한다.

(v_1, $?v_2$, $?_v$)($?v$, rdfs:subClassOf, v_3)

그러나 만일 v_1과 v_3이 모두 클래스 혹은 모두 인스턴스인 경우, 전이 프로퍼티와 owl:inverseOf를 이용하여 추론된 묵시적인 프로퍼티를 얻기 위해 다음 질의를 수행하게 된다. UNION 연산자를 기준으로 앞 부분은 전이 프로퍼티를 얻기 위한 질의이며 이후 질의는 묵시적인 프로퍼티를 얻기 위한 질의 부분이다. 전이 프로퍼티를 얻기 위한 이러한 연산은 중간 결과를 이용하여 반복적으로 수행된다. 따라서 초기에 주어진 트리플 t는 이와 같은 연산을 통해 확장되어 재정의된다.

(v_1, $?v_2$, $?v$)($?v$, $?v_2$, v_3)($?v_2$, rdf:type, owl:TransitiveProperty)

UNION (v_3, $?p$, v_1)($?p$, owl:inverseOf, $?v_2$)

이 논문에서는 ($?v_1$, $?v_2$, v_3) 및 (v_1, $?v_2$, $?v_3$)과 같은 Subject&Predicate-Variable 트리플 패턴이나 Predicate&Object-Variable 트리플 패턴에 대해서는 다루지 않는다. 이러한 질의에 대한 재작성은 다른 트리플 패턴에 비해 높은 복잡도와 비용을 요구하며, 무엇보다 이러한 질의 패턴은 사용자 질의에 거의 이용되지 않기 때문이다.

논문에서 질의 재작성을 위한 규칙 생성 과정은 후향

추론과 유사한 방식으로 이루어진다. 그러나 이는 단순히 질의 재작성에 필요한 규칙을 생성하기 위해 이용되며, 실제 질의에 대한 처리 과정에서는 트리플 매핑 연산을 통해 결과를 생성한다. 따라서 전통적인 인공지능에서의 후향 추론 방식과 다른 방식으로 질의 결과를 생성한다.

6. 실험 및 평가

이 장에서는 실험 및 평가 결과에 대하여 기술한다. 실험을 위한 프로토타입 시스템의 구조에 대하여 기술하고 실험을 위한 물리적 환경, 실험 데이타 셋, 테스트 질의 등에 대하여 기술한다. 마지막으로 정의한 환경하에서 수행한 실험 결과에 대하여 서술한다.

6.1 프로토타입 시스템 구현

질의 재작성 성능에 대한 실험을 위해 프로토타입 시스템을 구현하며 구현된 시스템의 전체적인 동작 과정 및 주요 컴포넌트는 그림 4와 같다.

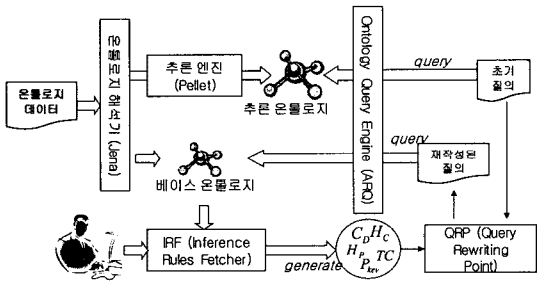

그림 4 전체적인 시스템 구조 및 동작 과정

추론 모델은 Pellet 추론 엔진을 통해서 생성되며 베이스 모델은 Jena를 통해서 생성한다. IRF(Inference Rules Fetcher)는 다양한 추론 규칙을 생성하여 XML 파일로 저장한다. 온톨로지에 대한 수정 연산이 수행될 때, IRF는 추론 규칙에 대한 수정을 동시에 수행한다. QRP(Query Rewriting Point)는 XML 파일로부터 추론 규칙을 판독하여 메모리에 로딩시킨다. 사용자가 SPARQL 질의를 입력하면, 결과를 얻기 위해 베이스 그래프 패턴을 추론 모델에 전달한다. QRP는 주어진 트리플 패턴이 임의의 추론 규칙과 일치하는지를 조사하기 위해 그래프 패턴을 획득한다. 트리플 패턴은 5장에서 기술한 연산들을 통해 재작성된다.

6.2 실험 환경

6.2.1 평가 항목 및 실험 데이타

이 논문의 목적은 추론 온톨로지 모델 기반의 접근 방법의 문제점을 해결하기 위한 방법으로 베이스 온톨로지 모델을 이용하여 SPARQL 질의에 대한 결과를 검색할 수 있는 알고리즘 개발에 있다. 따라서 평가는 기존의 접근 방법, 즉 추론 온톨로지 모델에 대한 질의 처리를 통해 결과를 반환해 주는 접근 방법의 대표적인 문제점인 비효율적인 메모리 사용 및 높은 로딩 시간에 초점을 둔다.

따라서 비교 평가 항목은 다음과 같다.

- 메모리 공간 크기: 메모리 상의 베이스 온톨로지 크기와 추론 온톨로지 크기
- 메모리 로딩 시간: 베이스 온톨로지와 추론 온톨로지 각각의 메모리 로딩 시간
- 질의 응답 시간 (질의 처리 성능): 추론 온톨로지 기반 접근 방법과 제안 방법 간의 질의 처리 시간

실험을 위한 데이타 셋은 Lehigh University에서 자체 개발한 저장소인 DLDB-OWL에 대한 성능 평가를 위해 개발한 온톨로지(LUBM, Lehigh Univ-Benchmark)를 이용한다[20]. 온톨로지 데이타 셋은 LUBM(n,s) 형식으로 표현할 수 있으며, n은 대학의 개수를 s는 시드 값을 각각 나타낸다. 따라서 n의 값이 증가할 수록 생성되는 데이타 셋의 사이즈는 증가하게 된다. 실험에 이용한 LUBM 데이타 셋의 크기는 다음과 같다.

- 클래스 인스턴스 개수: 20,659
- 프로퍼티 인스턴스 개수: 82,415

이 데이타 셋은 다음과 같은 추론 규칙들을 포함한다.

- CD = (a, headOf, b)(b, rdf:type,Department) → (a, rdf:type,Chair)
- HC = { {Thing[0,∞], Person[1, 4],Research Assistant[2, 3]}, {Thing[0,∞], Employee[1, 28], Faculty[2, 21], Lecturer[3, 4], . . .}, . . .}
- HP = { {memberOf[1, 6], worksFor[2, 5], headOf[3, 4]}, {degreeFrom[1, 8], undergraduateDegreeFrom[2, 3], masterDegreeFrom[4, 5], doctoralDegreeFrom[6, 7]}}

6.2.2 시스템 환경 및 SPARQL 테스트 질의

실험을 위한 물리적인 시스템 환경은 다음과 같다.

- CPU : 2.66GHz Pentium D
- 메모리 : 1GB
- 개발 언어 : Java SDK 1.5
- 온톨로지 저장소 : Jena 저장소

- 추론 온톨로지 생성기 및 추론기 : Jena API 및 추론 엔진

Jena는 온톨로지 생성, 편집, 저장 및 추론 등을 위한 다양한 API 및 도구를 제공한다. 오픈 API를 제공하기 때문에 시맨틱 웹을 위한 많은 응용 시스템 및 도구 개발에 활용되고 있다. 이 논문에서도 이러한 Jena의 특징과 장점을 활용하여 프로토타입을 개발하고 이를 이용하여 실험을 수행한다.

성능 평가 실험을 위해 테스트 질의가 요구되며, 이 논문에서는 DLDB 저장소 성능을 평가하기 위해 정의된 SPARQL 질의를 이용한다[20]. 질의는 총 14개로 구성되어 있으며 다음은 [20]에 정의된 질의 중 일부를 보여준다.

질의 예제 1. 대학원생들 중에서 GraduateCourse0 과목을 수강하는 학생을 검색하는 질의

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X
WHERE {?X rdf:type ub:GraduateStudent .
       ?X ub:takesCourse
       http://www.Department0.University0.edu/GraduateCourse0}
```

질의 예제 2. 모든 대원생들에 대해서 학사 학위를 취득한 대학교와 학과를 검색하는 질의

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX ub: <http://www.lehigh.edu/~zhp2/2004/0401/univ-bench.owl#>
SELECT ?X, ?Y, ?Z WHERE
       {?X rdf:type ub:GraduateStudent .
       ?Y rdf:type ub:University .
       ?Z rdf:type ub:Department .
       ?X ub:memberOf ?Z .
       ?Z ub:subOrganizationOf ?Y .
       ?X ub:undergraduateDegreeFrom ?Y}
       (type GraduateStudent ?X)
       (takesCourse ?X http://www.Department0.
       University0.edu/GraduateCourse0)
```

6.3 실험 결과 및 평가

그림 5(a)는 주어진 OWL 파일(웹 온톨로지)을 메모리 상에 로딩하는 소요되는 시간을 보여준다. 그림에서, 베이스 온톨로지 모델을 로딩하는 시간은 7,875(ms)인데 반해 추론 온톨로지 모델을 로딩하는데 소요되는 시간은 38,390(ms)이다. 추론 온톨로지 모델의 로딩 시간이 베이스 온톨로지 모델에 비해 약 5배 이상의 시간을 요구함을 알 수 있다.

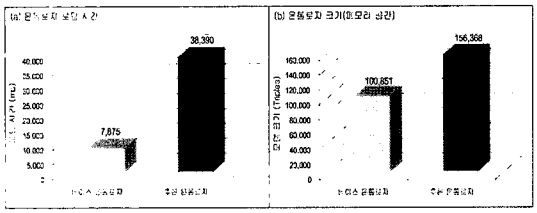

그림 5 온톨로지 로딩 시간 및 로딩된 온톨로지 크기 (메모리 공간 사용량)

그림 5(b)는 베이스 온톨로지 모델과 추론 온톨로지 모델의 메모리 공간 사용량, 즉 로딩된 온톨로지 사이즈를 보여준다. 그림에서, 베이스 온톨로지 모델은 100,851개의 트리플을 위한 공간이 요구되는 반면, 추론 온톨로지 모델은 156,368개의 트리플을 저장하기 위한 공간이 요구된다. 따라서 베이스 온톨로지 모델과 추론 온톨로지 모델 간 요구되는 공간 비율은 약 1.0:1.6이 된다. 즉, 추론 온톨로지 모델이 약 1.6배의 메모리 공간을 요구함을 알 수 있다.

표 1은 14개의 질의를 이용한 질의 처리 성능에 대한 실험 결과를 보여준다. 첫 번째 열은 [20]에서 정의한 14개의 질의의 순차 번호를 의미하며 두 번째 열은 재작성되지 않은 원래의 질의에 대한 응답 시간을 나타낸다. 마지막으로 세 번째 열은 이 논문에서 제안한 재작성 알고리즘에 의해 재작성 된 질의에 대한 응답 시간을 보여준다. 실험에서, 별표 심볼(*)로 표시된 부분은 결과를 반환하지 않는 경우를 나타낸다.

실험 결과에서, 일부 질의의 경우에는 재작성 질의의 처리 성능이 우수하였으나 일부는 원래 주어진 질의에 대한 질의 응답 시간이 우수함을 알 수 있다. 예를 들어, 질의 1-3, 12, 14의 경우에는 재작성 질의가 나은 성능을 보였으나 그 외는 원래의 질의가 보다 나은 성능을 보였다.

결론적으로, 메모리 크기, 로딩 시간 측면에서는 재작성 알고리즘을 이용한 SPARQL 질의 처리 방법이 추론 온톨로지 모델 기반 질의 처리 방법에 비해 우수하며, 질의 처리 성능 측면에서는 질의의 특성에 따라 선택적으로 접근 방법을 이용해야 할 것이다. 예를 들어, 온톨로지 모델의 독립성을 유지하면서 잦은 로딩이 요

표 1 질의 처리 성능에 대한 실험 결과 (단위: ms)

질의	원래의 질의 처리 시간	재작성 질의 처리 시간
1	1,094	265
2	*	78
3	3,312	32
4	375	1,250
5	4,719	40,515
6	312	312
7	6,677,203	*
8	625,172	*
9	*	*
10	4,079	8,781
11	296	890
12	1,297	328
13	3,844	53,266
14	359	172

구되는 경우나, 메모리 크기에 대한 제약성을 지니는 경우에는 재작성 알고리즘을 이용한 SPARQL 질의 처리 기법이 선택되어야 한다. 또는 1-3, 12 및 14와 같은 질의 특성을 지니는 처리 시스템의 경우에도 제안한 방법이 채용되어야 한다. 그러나 일부 제안한 접근 방법의 성능이 저하되는 질의와 같은 특성을 지니는 질의가 요구되고 로딩 시간과의 다면 성능 분석 결과가 우수한 경우에는 기존 접근 방법이 선택되어야 할 것이다. 그러나 이 경우, 메모리 크기에 대한 제약성이 없다는 조건이 만족되어야만 한다.

7. 결론 및 향후 연구

이 논문에서는 SPARQL 추론 질의 처리를 위해 추론 온톨로지 모델을 생성하여 이용하는 기존 접근 방법과 달리, 베이스 온톨로지 모델을 이용하여 SPARQL 질의를 재작성하여 결과를 생성하는 방법을 제안하였다. 재작성 알고리즘을 정의하였고 이에 대한 프로토타입 시스템을 개발하여 실험 평가를 수행하였다.

구조적인 특징 및 실험 결과에서, 제안한 접근 방법이 추론 온톨로지 모델을 생성하고 관리함에 있어 낮은 비용을 요구함을 알 수 있다. 또한 베이스 온톨로지 모델을 여러 사용자에게 그대로 재사용할 수 있다는 장점을 지닌다. 따라서 관리 및 활용의 편의성은 물론 메모리 공간 및 온톨로지 로딩 측면에서 보다 효율적이다. 또한 질의 처리 성능 측면에서도 기존 접근 방법과 비교하여 유사한 성능을 제공한다.

결론적으로, 제안한 재작성 알고리즘을 이용한 SPARQL 질의 처리 접근 방법은 다음과 같은 장점을 지닌다.

- 효율적인 온톨로지 로딩
- 효율적인 메모리 사용
- 온톨로지 추론 규칙에 독립적인 온톨로지 모델 재사용
- 메모리 크기가 제한적인 디바이스 기반의 서비스 개발에 적합
- 성능이 제한적인 환경에 적합한 온톨로지 로딩 및 활용

기존 접근 방법과 비교하여, 제안한 재작성 알고리즘 기반의 SPARQL 추론 질의 처리 방법이 여러 장점을 제공하지만, 여전히 몇 가지 개선해야 할 사항을 지닌다. 먼저 실험 평가에서 보다 방대한 크기의 온톨로지를 이용한 실험이 요구되며, 또한 질의의 특성에 따라 접근 방법을 선택할 수 있는 기준에 대한 연구가 이루어져야 한다. 마지막으로, 재작성 질의의 처리 성능을 개선할 수 있는 최적화 알고리즘의 개발이 요구된다.

참 고 문 헌

[1] Klyne, G. and Carroll, J.J., "Resource Description Framework (RDF): Concepts and Abstract Syntax," W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-concepts-20040210/

[2] Patel-Schneider, P.F., Hayes, P., and Horrocks, I., "OWL Web Ontology Language Semantics and Abstract Syntax," W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-owl-semantics-20040210/

[3] Jena - A Semantic Web Framework for Java. http://jena.sourceforge.net

[4] Broekstra, J., Kampman, A., and Harmelen, F.v., "Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema," Springer Verlag, Lecture Notes in Computer Sciences, Vol. 2342, pp 54 - 68, 2002.

[5] Hayes, P., "RDF Semantics," W3C Recommendation, 10 February 2004. http://www.w3.org/TR/2004/REC-rdf-mt-20040210/

[6] Horrocks, I. and Sattler, U., "A Tableaux Decision Procedure for SHOIQ," In Proceedings of the 19th International Joint Conference on Artificial Intelligence, pp 448-453, 2005.

[7] Broekstra, J., Kampman, A. "Exploring a naive practical approach," In Workshop on Practical and Scalable Semantic Systems at the Second International Semantic Web Conference, Sanibel Island, Florida, October 2003.

[8] Noy, N.F. and Klein, M., "Ontology Evolution: Not the Same as Schema Evolution," Knowledge and Information Systems, Vol. 6, pp. 428-440, 2004.

[9] Pinto, H.S. and Martins, J.P., "Ontologies: How can They be Built?," Knowledge and Information Systems, Vol. 6, pp. 441-464, 2004.

[10] Kotis, K. and Vouros, G.A., "Human-centered ontology engineering: The HCOME methodology,"

Knowledge and Information Systems, Vol. No. 1, pp. 109-131, 2006.
[11] Fuxman, A. and Miller, J., "First-Order Query Rewriting for Inconsistent Databases," In Proceedings of the 10th International Conference on Database Theory, Scotland, pp. 337-351, 2005.
[12] Vidal, M.E., Raschid, L., Marquez, N., Cardenas, M., and Wu, Y., "Query Rewriting in the Semantic Web," In Proceedings of the 22nd International Conference on Data Engineering Workshops, USA, 2006.
[13] Halevy A.Y., "Answering Queries Using Views: A survey," VLDB Journal: Very Large Data Bases, Vol. 10, No. 4, pp. 270-294, 2001.
[14] Prud'hommeaux, E. and Seaborne, A., "SPARQL Query Language for RDF," W3C Working Draft, 12 October 2004. http://www.w3.org/TR/2004/WD-rdf-sparql-query-20041012/
[15] Seaborne, A., "RDQL - A query language for RDF," W3C Member Submission, 9 January 2004. http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/
[16] Fikes, R., Hayes, P., and Horrocks, I. "OWL-QL - A Language for Deductive Query Answering on the Semantic Web," Technical Report KSL-03-14, Stanford University, CA, 2003.
[17] Volker, H., Moller, R., and Wessel, M., "Querying the Semantic Web with Racer + nRQL," In Proceedings of the KI-2004 International Workshop on Applications of Description Logics, Ulm, Germany, September 24, 2004.
[18] Perez, J., Arenas, M., and Gutierrez, C., "Semantics and Complexity of SPARQL," Springer Verlag, In Proceedings of the 5th International Semantic Web Conference, Vol. 4273, pp 30 - 43, 2006.
[19] Zhang, C., Naughton, J., DeWitt, D., Luo, Q, and Lohman G., "On Supporting Containment Queries in Relational Database Management Systems," In Proceedings of the 2001 ACM SIGMOD Conference, 2001.
[20] Guo, Y., Pan, Z., and Heflin, J., "LUBM: A Benchmark for OWL Knowledge Base Systems," Journal of Web Semantics, Vol. 3, No. 2, pp. 158-182, 2005.

정 동 원
정보과학회논문지 : 데이타베이스
제 35권 제 1 호 참조

Yixin Jing
정보과학회논문지 : 데이타베이스
제 35 권 제 1 호 참조

백 두 권
정보과학회논문지 : 데이타베이스
제 35 권 제 1 호 참조