RDF 스키마 함의 규칙 적용 순서를 이용한 RDFS 추론 엔진의 최적화

(An Optimization Technique for RDFS Inference the Applied Order of RDF Schema Entailment Rules)

김 기 성 † 유 상 원 † 이 태 휘 † 김 형 주 ††
(Kisung Kim) (Sangwon Yoo) (Taewhi Lee) (Hyung-Joo Kim)

요 약 W3C의 권고안인 RDF Semantics는 RDFS 추론에 사용할 RDFS 함의 규칙을 제안하였다. 널리 사용되고 있는 RDF 저장소 시스템인 Sesame는 전방향 추론 방식을 사용하여 RDBMS 기반 RDFS 추론을 지원한다. Sesame의 전방향 추론 전략을 사용할 때에는 데이타 저장 시에 추론을 하기 때문에 추론 성능이 데이타 저장 성능에 영향을 미친다. 이런 문제점을 개선하기 위해 본 논문에서는 RDBMS 기반의 전방향 추론 엔진의 성능 향상을 위한 RDFS 함의 규칙 적용 순서를 제안한다. 제안한 규칙 적용 순서는 추론 과정을 대부분의 경우 추론 과정의 반복 없이 한번에 끝낼 수 있도록 하며 완벽한 추론 결과를 보장한다. 또한 앞서 적용한 규칙에 의해 생성된 결과를 추측할 수 있어 추론 과정에서 중복된 결과 생성을 줄일 수 있다. 본 논문에서는 실제 사용하는 RDF 데이타들을 사용하여 Sesame와의 추론 성능을 비교하며 제안한 방법이 RDFS 추론 성능을 향상시킬 수 있음을 보인다.

키워드 : 시멘틱 웹, RDF, 규칙 기반 추론, 전방향 추론

Abstract RDF Semantics, one of W3C Recommendations, provides the RDFS entailment rules, which are used for the RDFS inference. Sesame, which is well known RDF repository, supports the RDBMS-based RDFS inference using the forward-chaining strategy. Since inferencing in the forward-chaining strategy is performed in the data loading time, the data loading time in Sesame is slow down be inferencing. In this paper, we propose the order scheme for applying the RDFS entailment rules to improve inference performance. The proposed application order makes the inference process terminate without repetition of the process for most cases and guarantees the completeness of inference result. Also the application order helps to reduce redundant results during the inference by predicting the results which were made already by previously applied rules. In this paper, we show that our approaches can improve the inference performance with comparisons to the original Sesame using several real-life RDF datasets.

Key words : Semantic Web, RDF, Rule-based Inference, Forward-chaining

1. 서 론

Resource Description Framework(RDF)[1]은 웹 상의 리소스에 대한 메타데이타를 기술하는 표준으로서 W3C(World Wide Web Consortium)에서 제정하였다. RDF 데이타는 주어(subject), 술어(predicate), 목적어(object)의 삼형식(triple) 구조를 갖는 문장(statement)의 집합으로 볼 수 있다. RDF Schema(RDFS)[2]는 이런 RDF 데이타 모델에 부가적인 의미 정보를 기술할 수 있는 방법을 제공한다. 부가적인 의미 정보란 구체적으로 리소스의 타입, 클래스와 프로퍼티의 계층 관계를 말한다. RDFS 추론이란 이런 RDFS 정보를 이용하여 RDF 데이타에 기술 되지 않은 새로운 문장을 추가하는 과정으로 볼 수 있다.

추론 과정은 기존의 지식 기반 시스템(Knowledge

· 본 연구는 BK-21 정보기술 사업단과 정보통신부 및 정보통신연구진흥원의 대학 IT연구센터 육성 지원 사업(IITA-2005-C1090-0502-0016)의 연구결과로 수행되었음
† 학생회원 : 서울대학교 컴퓨터공학부
kskim@oopsla.snu.ac.kr
twlee@oopsla.snu.ac.kr
: swyoo@oopsla.snu.ac.kr
†† 종신회원 서울대학교 컴퓨터공학부 교수
hjk@oopsla.snu.ac.kr
논문접수 : 2005년 8월 1일
심사완료 : 2005년 12월 11일

Base System)에서 사용한 규칙 기반 추론을 사용한다. W3C의 권고안인 RDF Semantics[3]에서는 RDFS 추론에 사용될 13가지 RDFS 함의 규칙을 제안하였으며 이 규칙을 통해 모든 RDFS 추론 결과를 얻을 수 있음을 보였다. 현재 여러 RDF 저장소 시스템들이 이 RDFS 함의 규칙을 사용하여 RDFS 추론을 지원하고 있다. RDFS 추론 전략은 추론 방향에 따라 크게 전방향 추론(forward chaining)과 후방향 추론(backward chaining)으로 나뉜다. 전방향 추론은 현재 정의된 사실들에 적용 가능한 규칙을 연속적으로 적용하여 최종 목표에 해당하는 도달하는 방식을 말한다. 반면 후방향 추론은 찾고자 하는 목표에서 시작하여 이를 지지할 수 있는 사실들을 찾는 방식을 말한다. 보통 전방향 추론 방식은 추론의 결과를 미리 저장을 하는 즉시 평가(eager evaluation)에서 사용하며 후방향 추론 방식은 질의 처리 시간에 추론을 하는 지연 평가(lazy evaluation)에서 사용한다.

대표적인 RDF 저장소 시스템인 Sesame[4]는 RDBMS에 기반한 영속적 데이타 모델(persistent data model)과 함께 RDBMS 기반의 RDFS 추론엔진을 제공한다. Sesame의 RDFS 추론 전략은 즉시 평가 방식을 사용하며 전방향 추론을 사용한다. 즉 데이타 저장 과정에서 RDFS 추론 과정을 거치고 추론 결과를 모두 저장해둔다. 이런 전방향 추론 전략은 모든 추론 결과를 저장하기 때문에 질의 처리 속도가 빠르다는 장점이 있다. 또한 질의 처리 구현 자체가 매우 간단해 진다. 이런 장점이 있는 반면 다음과 같은 성능의 문제를 발생시킨다. 우선, 데이타 저장시 추론을 하기 때문에 RDFS 추론의 성능이 데이타 저장 성능에 큰 영향을 미친다. NCI Ontology[1] 데이타의 경우 데이타 저장 시간의 약 30%가 추론 과정에서 소요된다. 추론 성능이 데이타 저장 성능에만 영향을 미치는 것은 아니다. [5]에서는 실체화된 온톨로지의 업데이트시의 유지를 위한 방법을 제안하였는데, 이때 온톨로지의 변화를 적용하기 위한 과정에는 기본적으로 추론 과정이 포함된다. 즉, 데이타 업데이트시에도 추론 과정이 필요하다. 따라서 효율적인 RDFS 데이타의 관리를 위해서는 추론 성능의 개선이 필요하다.

Sesame의 추론 과정은 13개의 RDFS 함의 규칙을 반복적으로 적용하여 더 이상 추론 된 문장이 없을 때 추론이 종료된다. 이렇게 더 이상 결과를 얻을 수 없을 때까지 반복적으로 규칙을 적용하기 때문에 이를 고갈적 전방향 추론(exhaustive forward chaining)이라고 한다[6]. 고갈적 전방향 추론은 크게 두 가지 문제점을 갖는다. 우선 규칙을 반복적으로 적용할 때 모든 규칙이 결과를 만들어 내지는 않는다. 즉 불필요한 규칙의 적용이 생길 수 있다. 또한 추론 과정 도중 중복된 결과가 생성된다[7]. 중복된 결과가 빈번하게 생성 되면 전체적인 추론 성능이 떨어지게 된다. Sesame에서는 불필요한 규칙의 적용을 줄이기 위해 규칙간의 의존성 관계를 사용하였다. 그러나 규칙간의 의존성만으로는 모든 비효율성을 제거하지는 못한다. 모든 불필요한 규칙 적용을 제거 하지 못하며, 중복된 결과 생성에 대한 고려가 없기 때문이다.

최근의 RDF 저장소 시스템의 성능 비교 논문[8]에서 Sesame는 데이타 저장 성능의 확장성(scalability)에 문제점이 있는 것으로 알려졌다. 이 문제의 원인 중 하나로 추론 방식이 거론되고 있다. 따라서 데이타 저장 성능의 확장성을 위해서는 추론 방식의 개선이 필요하다. 본 논문에서는 RDFS 함의 규칙의 적용 순서를 정하여 불필요한 규칙의 적용을 최소화하고, 중복된 결과의 생성을 줄이는 방법을 제안하고자 한다.

논문의 구성은 다음과 같다. 우선 2장 관련 연구에서는 여러 RDF 저장소 시스템의 RDFS 추론 지원을 살펴보고 3장에서는 RDFS 추론에 대한 배경 지식을 설명한다. 4장에서는 Sesame에서의 추론 과정의 문제점과 우리가 제안하는 개선 사항에 대해 설명하고 5장에서 실험 결과에 대해 논한 뒤 마지막으로 6장에서 결론과 향후 연구에 대해 기술한다.

2. 관련 연구

RDF Semantics[3]는 W3C 권고안으로 모델 이론(Model Theory)을 제안하였다. 이 모델 이론은 RDF와 RDFS의 이론적 기반을 제공하는 모델로서, 초기 RDF, RDFS의 의미적 모호성을 정리하였다고 할 수 있다. 또한 모델 이론에서는 RDFS 추론에 사용될 여러 함의 규칙을 제안하고 있다. WILBUR[7]는 지연 평가 전략을 사용하여 RDFS 추론을 지원하는 시스템으로 RDFS 함의 규칙이 매우 중복적이기 때문에 즉시 평가를 사용하여 이 규칙을 반복적으로 적용하는 것은 현실적인 방법이 되지 못한다고 주장하였다. Jena[9]는 혼합형 방식을 취하여 두 방식의 단점을 보안하려 하였으나 Jena는 현재 RDBMS 기반의 추론 엔진은 지원하지 않고 있다.

Sesame는 규칙간의 의존성 관계를 사용하여 전방향 추론의 성능을 개선하였으며 전방향 추론 방식이 매우 간단하며 실용적인 방법임을 보였다[6]. 즉시평가를 사용하는 Sesame는 질의 성능이 우수한 것으로 평가되며[8] 즉시 평가 방식이 RDFS 추론 방식으로 유용함을 보였다. 그러나 Sesame의 전방향 추론 방식은 아직 추론 성능에 있어서 데이타 저장 속도, 확장성 등의 문제

1) http://www.mindswap.org/2003/CancerOntology/

점을 갖고 있어 전방향 추론의 최적화를 필요로 하고 있다.

전통적으로 전방향 추론을 처리하는 데에는 Rete 알고리즘[10]이 가장 효율적인 알고리즘으로 평가 받아왔다. Rete 알고리즘은 추론 규칙들간의 관계를 고려한 네트워크로 생성하고 네트워크의 흐름에 따라 추론을 하게 된다. 이때에 부분 매칭이 일어나는 규칙에 대해서는 중간 결과를 저장함으로써 중복된 추론 과정을 제거하여 추론 성능을 향상시킨다. 그러나 Rete 알고리즘에서는 중간 결과를 모두 메모리상에서 저장하는 것을 가정하고 있기 때문에 본 논문에서 가정하는 대용량 RDF 데이타에 대한 추론에 응용하기 위해서는 별도의 조치가 필요하다. 예를 들어 RDFS 함의 규칙 중 rdfs2와 같은 규칙에서는 전제에 해당하는 패턴이 모든 문장과 매칭된다. 따라서 현재 저장하려는 RDF 문서의 모든 문장이 중간결과에 해당하기 때문에 중간결과의 크기가 매우 커지게 된다. RDBMS에 기반한 Rete 알고리즘의 활용은 별도의 연구가 필요할 것으로 생각된다.

본 논문에서는 RDBMS에 기반한 전방향 추론 방식을 사용할 때의 최적화 방안에 대해 논의하며 Sesame 시스템과의 성능 비교를 통해 우리가 제안한 방법의 성능을 평가하였다.

3. 배경 지식

3.1 RDFS 함의 규칙

RDF Semantics에서 제안한 RDFS 함의 규칙은 기본적으로 다른 문장의 존재로부터 다른 문장을 추가하는 구조를 갖는다. 본 논문에서 고려하는 함의 규칙은 RDF Semantics에서 제안한 RDF 함의 규칙 1번과 RDFS 함의 규칙 2~13번까지이다. 이 규칙들은 Sesame 1.1.3의 RDFS 추론엔진에서 사용한 함의 규칙들이다. 실제로 RDF Semantics에서는 이외에 확장 함의 규칙(extensional entailment rules), 데이타타입 함의 규칙(datatype entailment rules)을 제공하고 있으나 본 논문에서는 이들 규칙은 제외하였다. 실제로 현재 여러 시스템에서도 이들 추가적인 규칙에 대해서는 고려하지 않는 경우가 많다. 표 1[3]은 본 논문에서 고려한 규칙들을 나타낸다.

각 규칙은 전제와 결론으로 나뉜다. 각 규칙은 다음과 같이 해석한다.

현재 저장소에 전제의 패턴에 해당하는 문장이 존재하면 결론의 문장을 저장소에 추가하라.

표 1에서 uuu, aaa 등은 리소스의 URI를 나타내는 변수이다. 각 규칙은 현재 저장소에 저장된 문장에 기반하여 RDFS 의미론에 맞는 새로운 문장을 추가 한다. 예를 들어 rdfs2는 프로퍼티의 도메인 정보를 이용한 리소스의 타입을 추론하는 규칙이다. 이 규칙의 의미는 다음과 같다.

프로퍼티 aaa의 도메인이 xxx로 정의되어 있다면,
aaa가 술어로 사용된 문장의 주어(uuu)의 타입은 xxx 이다.

3.2 RDFS 함의 규칙간의 의존 관계

표 1 RDFS 함의 규칙

규칙	전제	결론
rdf1	uuu aaa yyy	aaa rdf:type rdf:Property
rdfs2	aaa rdfs:domain xxx uuu aaa yyy	uuu rdf:type xxx
rdfs3	aaa rdfs:range xxx uuu aaa yyy	yyy rdf:type xxx
rdfs4a	uuu aaa yyy	uuu rdf:type rdfs:Resource
rdfs4b	uuu aaa yyy	yyy rdf:type rdfs:Resource
rdfs5	uuu rdfs:subPropertyOf vvv vvv rdfs:subPropertyOf xxx	uuu rdfs:subPropertyOf xxx
rdfs6	uuu rdf:type rdf:Property	uuu rdfs:subPropertyOf uuu
rdfs7	aaa rdfs:subPropertyOf bbb uuu aaa yyy	uuu bbb yyy
rdfs8	uuu rdf:type rdfs:Class	uuu rdfs:subClassOf rdfs:Resource
rdfs9	uuu rdfs:subClassOf xxx vvv rdf:type uuu	vvv rdf:type xxx
rdfs10	uuu rdf:type rdfs:Class	uuu rdfs:subClassOf uuu
rdfs11	uuu rdfs:subClassOf vvv vvv rdfs:subClassOf xxx	uuu rdfs:subPropertyOf xxx
rdfs12	uuu rdf:type rdfs:ContainMembershopProperty	uuu rdfs:subPropertyOf rdfs:member
rdfs13	uuu rdf:type rdfs:Datatype	uuu rdfs:subClassOf rdfs:Literal

한 함의 규칙을 적용하여 생긴 문장이 다른 함의 규칙의 전제 패턴과 매치 될 수 있다. 이와 같이 두 함의 규칙 $r_1 : \phi_1 \rightarrow \psi_1$, $r_2 : \phi_2 \rightarrow \psi_2$에 대해, 결론 $s \in \psi_1$이 전제 $p \in \phi_2$와 대응이 되면 r_1은 r_2에 의존적이라고 한다. 이때 두 규칙의 의존 관계를 다음과 같이 나타낸다.

$$r_1 \rightarrow r_2$$

예를 들어, rdfs2의 결론은 다음과 같다.

uuu rdf:type xxx

rdfs2를 적용한 결과 생성되는 문장은 rdfs6의 전제인

uuu rdf:type rdfs:Property

의 패턴을 가질 수 있다. 다시 말해 rdfs2의 적용 결과 생긴 문장으로 인해 rdfs6을 적용하여 새로운 문장을 만들 수 있는 가능성이 생긴 것이다. 이런 경우 rdfs2를 적용한 후 rdfs6을 반드시 적용해야 완벽한 추론 결과를 얻을 수 있다. 이렇게 한 규칙의 적용이 다른 규칙의 적용을 야기하는 것을 트리거(trigger)한다고 말한다. 표 2[6]는 13개의 RDFS 함의 규칙간의 의존 관계를 정리한 표이다.[2)]

표 2에서 X 표시가 된 셀은 행의 규칙이 열에 해당하는 규칙에 의존적이라는 의미이다. 이 표에서 rdfs2가 rdfs6에 의존적임을 알 수 있다. 표에서 rdfs2, rdfs3, rdfs5, rdfs7, rdfs9, rdfs11은 각각 rdfsX_1, rdfsX_2와 같이 두 개의 규칙으로 나뉘었다. 이 규칙들은 전제가 두 개의 문장 패턴으로 구성된 규칙들로 이들을 나눈 이유는 Sesame에서 규칙 적용 과정을 구현한 방식과 관련이 있다. 이는 다음 장에서 자세히 설명하도록 하겠다.

4. RDFS 추론의 최적화 기법

이번 장에서는 우리가 제안하는 RDFS 추론의 최적화 기법에 대해 기술한다. 먼저 RDBMS 기반의 RDFS 추론에 대해 설명한 뒤 함의 규칙간 의존성을 고려한 기법이 갖는 문제점과 우리가 제안한 최적화 방법에서 그 문제점들을 어떻게 해결하였는지 설명하겠다.

4.1 RDBMS 기반의 RDFS 추론

현재 일반적인 RDBMS 기반의 RDF 데이타 모델은 triples 테이블이 중심을 이룬다. triples 테이블은 RDF 데이타의 모든 문장을 저장하기 위한 테이블이다. 이 triples 테이블은 각 저장소의 구현 방식에 따라 차이는 있지만 그림 1과 같은 스키마를 갖는다. 테이블의 각 열이 갖는 의미는 다음과 같다. ID 열은 문장 고유의 ID를 저장하는 열이며 Subject, Predicate, Object 열은 한 문장을 주어, 술어, 목적어로 분해하여 각각을 저장하는 열이다. Explicit 열은 이 문장이 RDF 데이타에 명시적으로 정의되었는지의 여부를 나타내는 것으로 추론에 의해 생성된 문장은 0의 값을 갖고 데이타에 명시적으로 정의된 문장은 1의 값을 갖는다[4]. triples 테이

2) 표 2는 [6]의 의존성 테이블과 차이가 있다. 이는 [6]의 의존성 테이블은 2003년 당시의 working draft 상태의 RDF Semantics 문서에서 제안한 함의 규칙에 대해 고려한 것이기 때문이다. 표 2는 Sesame 1.1.3에 반영되어 있는 내용을 인용한 것으로 2004년에 권고안으로 제정된 RDF Semantics 문서에서 제안한 함의 규칙에 대한 의존성 테이블이다.

표 2 RDFS 함의 규칙간 의존성

규칙	1	2_1	2_2	3_1	3_2	4a	4b	5_1	5_2	6	7_1	7_2	8	9_1	9_2	10	11_1	11_2	12	13
1	-	X	-	X	-	X	-	-	-	X	X	-	-	X	-	-	-	-	-	-
2 1	-	X	-	X	-	-	-	-	-	X	X	-	X	X	-	X	-	-	X	X
2 2	-	X	-	X	-	-	-	-	-	X	X	-	X	X	-	X	-	-	X	X
3 1	-	X	-	X	-	-	-	-	-	X	X	-	X	X	-	X	-	-	X	X
3 2	-	X	-	X	-	-	-	-	-	X	X	-	X	X	-	X	-	-	X	X
4a	-	X	-	X	-	-	-	-	-	-	X	-	-	X	-	-	-	-	-	-
4b	-	X	-	X	-	-	-	-	-	-	X	-	-	X	-	-	-	-	-	-
5 1	-	-	-	-	-	-	-	X	X	-	X	X	-	-	-	-	-	-	-	-
5 2	-	-	-	-	-	-	-	X	X	-	X	X	-	-	-	-	-	-	-	-
6	-	X	-	X	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-
7 1	-	X	X	X	X	-	-	X	X	X	X	X	X	X	X	X	X	X	X	X
7 2	-	X	X	X	X	-	-	X	X	X	X	X	X	X	X	X	X	X	X	X
8	-	X	-	X	-	-	-	-	-	-	X	-	-	-	X	-	X	X	-	-
9 1	-	-	-	X	-	-	-	-	-	X	X	-	X	X	-	X	-	-	X	X
9 2	-	-	-	X	-	-	-	-	-	X	X	-	X	X	-	X	-	-	X	X
10	-	X	-	X	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-
11 1	-	-	-	-	-	-	-	-	-	-	X	-	-	-	X	-	X	X	-	-
11 2	-	-	-	-	-	-	-	-	-	-	X	-	-	-	X	-	X	X	-	-
12	-	X	-	X	-	-	-	X	X	-	X	X	-	-	-	-	-	-	-	-
13	-	X	-	X	-	-	-	-	-	-	X	-	-	-	X	-	X	X	-	-

ID	Subject	Predicate	Object	Explicit
1	write	rdfs:range	article	1
2	article	rdfs:subClassOf	publication	1
3	Jim	write	writing01	1

(a) 추론 전

ID	Subject	Predicate	Object	Explicit
1	write	rdfs:range	article	1
2	article	rdfs:subClassOf	publication	1
3	Jim	write	writing01	1
4	**writing01**	**rdf:type**	**article**	**0**
5	**writing01**	**rdf:type**	**publication**	**0**

(b) 추론 후

그림 1 추론 전과 후의 triples 테이블의 예

블은 중복된 문장을 갖지 않아야 한다. 그림 1은 추론 전과 추론 후의 triples 테이블의 모습을 나타낸다. 추론 후에는 explicit 열이 0인 문장이 생긴 것을 알 수 있다. 이 문장들은 RDFS 함의 규칙을 적용한 결과 생긴 것이다. 실제 추론의 결과에는 각 리소스들의 기본 타입을 지정하는 문장이 rdf1, rdfs4a, rdfs4b에 의해서 더 생기게 된다. 하지만 여기에서는 간략하게 나타내기 위해 기본 타입에 대한 추론 결과는 나타내지 않았다. 4번 문장은 1,3번 문장으로부터 rdfs3규칙에 의해 생성된다. 5번 문장은 2번과 4번 문장으로부터 rdfs9를 적용하여 생성하게 된다. 이때 주목할 점은 rdfs3에 의해 4번 문장이 생성된 후에야 rdfs9를 적용하여 5번 문장을 만들 수 있다는 것이다. 이는 rdfs3이 rdfs9에 의존적이기 때문이다. 이런 의존성 때문에 규칙을 반복적으로 적용해야 한다.

S_{old} = Set of statements already stored
S_{new} = Set of statements newly stored
$S_{old} = S_{old} \cup S_{new}$
Do until($S_{new} = \phi$)
  $S_{inferred} = \phi$
  //Application of rules with one premise
  for r ∈ {rdf1, rdfs4a, rdfs4b, rdfs6, rdfs8, rdfs10, rdfs12, rdfs13}
    $S_{inferred} = S_{inferred} \cup \text{applyRule}(r, S_{new})$
  //Application of rules with two premise
  for r ∈ {rdfs2, rdfs3, rdfs5, rdfs7, rdfs9, rdfs11}
    $S_{inferred} = S_{inferred} \cup \text{applyRule2}(rdfs2, S_{old}, S_{new})$
    $S_{inferred} = S_{inferred} \cup \text{applyRule2}(rdfs2, S_{new}, S_{old})$
  $S_{old} = S_{old} \cup S_{inferred}$
  $S_{new} = S_{inferred}$

그림 2 고갈적 전방향 추론 알고리즘

전체적인 추론 과정은 그림 2와 같다. 그림 2의 알고리즘에서는 규칙 적용 순서에 대해 어떠한 제약도 없음을 주목하자. 이미 저장되어 있는 문장의 집합을 S_{old}, 새로 저장된 문장의 집합을 S_{new}라 하자. $\text{applyRule}(r,S)$는 S에 규칙 r을 적용한 결과 생성된 문장의 집합이다. $\text{applyRule2}(r,S_1,S_2)$는 전제의 패턴이 두 개인 규칙을 적용한 결과 생성된 문장의 집합을 나타내는데, 두 인자 S_1, S_2는 두 개의 전제 패턴을 각각 어느 집합에서 찾을지를 지정해준다.

실제 구현에서 S_{old}는 triples 테이블을 나타내며, S_{new}는 새로 저장하는 문장을 갖고 있는 new_triples 테이블을 나타낸다. new_triples 테이블은 triples 테이블과 동일한 스키마를 갖는다. $\text{applyRule}(r,S)$은 triples 테이블에서 규칙 r의 패턴에 해당하는 문장을 SELECT하는 SQL 질의문으로 구현된다. $\text{applyRule2}(r,S_1,S_2)$는 두 개의 패턴을 전제로 갖는 규칙에 대해 적용되며 new_triples 테이블과 triples 테이블에 대한 조인 연산으로 구현된다. 이때 어느 패턴을 어느 테이블에서 갖고 오느냐에 대해 다음의 세가지 경우에 대해 생각을 해야 한다. 두 패턴을 p_1, p_2라고 하면,

1. p_1, p_2 모두 새로 입력된 문장에서 검색하는 경우
2. p_1은 새로 입력된 문장에서, p_2는 이미 저장된 문장에서 검색하는 경우
3. 위의 반대 경우

p_1, p_2 모두 이미 저장된 문장에서 검색하는 것은 이미 이전에 적용되었을 것이기 때문에 고려하지 않아도 된다. 이와 같이 세가지 경우가 존재하지만 triples 테이블이 새로 입력된 문장과 이전에 저장된 문장을 모두 갖고 있기 때문에 두 번의 규칙 적용으로 가능하다. 앞의 규칙 의존성 테이블에서 한 개의 규칙을 두 개로 나눈 것은 바로 이 두번의 규칙 적용을 의미한다.

4.2 함의 규칙간 의존성을 이용한 기법

앞에서 설명한 고갈적 전방향 추론을 그대로 사용할 경우 각 반복 마다 모든 규칙을 적용하기 때문에 단순한 데이타 저장 시에도 규칙 적용 횟수가 매우 많아 질 수 있다. 규칙 적용 횟수를 줄이기 위해 Sesame는 앞서 설명한 함의 규칙간의 의존성을 사용한다[6]. Sesame에서는 함의 규칙간 의존성을 사용하여 규칙을 적용하기 전에 규칙을 적용할 필요가 있는지에 대해 먼저 체크를 한다. 규칙 적용의 두 번째 반복에서부터는 이전 반복의 추론 결과로 인해 트리거 되는 규칙만을 적용한다. 다시 말해, n번째 반복에서는 n-1번째 반복에서 새로운 결과를 만든 규칙과 의존성이 있는 규칙만을 적용한다. 예를 들어, 이전 규칙 적용 과정에서 rdfs10

만이 새로운 결과를 만들었다면 다음 규칙 적용에는 모든 규칙을 적용할 필요 없이 rdfs2_1, rdfs3_1, rdfs7_1만을 적용하게 된다. 이 최적화는 불필요한 규칙의 적용을 감소하여 추론 과정을 효율적으로 만든다. 그러나 함의 규칙간의 의존성을 사용한 최적화만으로는 모든 비효율성을 제거하지는 못한다. 우선 불필요한 규칙의 적용이 여전히 존재한다. 불필요한 규칙의 적용은 다음과 같이 두 가지 경우로 볼 수 있다.

- 추이성(transitive property) 추론에 따른 규칙 적용의 반복: rdfs:subPropertyOf, rdfs:subClassOf 프로퍼티는 추이성을 갖는다. rdfs5, rdfs11은 이들 프로퍼티의 추이성 닫힌 집합(transitive closure)을 계산하는 규칙이다. 만일 클래스 또는 프로퍼티의 계층 구조의 깊이가 큰 데이타에 대한 추론을 한다면 이 규칙들을 모든 추이성 닫힌 집합을 모두 계산 할 때까지 반복적으로 적용해야 하기 때문에 여러 번 적용된다. 이때, 각 규칙은 각각 rdfs7_2, rdfs9_2를 트리거 한다. 따라서 매번 rdfs5, rdfs11을 적용할 때마다 이 규칙들을 적용해야 한다. 그러나 만일 모든 추이성 닫힌 집합이 계산 되어 있다면 이들 규칙들은 한번의 적용으로도 충분하다.
- RDFS의 기본 어휘의 확장: RDFS에는 rdfs:Class, rdfs:subClassOf등과 같이 미리 정의된 클래스, 프로퍼티가 존재한다. 저장하려는 RDF 데이타에 이러한 기본 클래스, 프로퍼티에 대해 하위 클래스, 프로퍼티 등이 정의 되어 있을 수도 있다. 표2의 함의 규칙의 의존성에는 이러한 경우를 고려한 의존 관계가 많다. 예를 들어 rdfs7_1이 rdfs2_2를 트리거하는 이유는 rdfs7_1의 결과로 rdfs:domain의 하위 프로퍼티가 생겼을 경우를 고려하였기 때문이다. 그러나 실제로 이러한 경우는 빈번하지 않기 때문에 무조건 트리거 하기 보다는 이런 결과가 있을 때만 트리거하는 것이 효율적이다.

고갈적 전방향 추론의 또 다른 문제점은 추론 도중 중복된 결과를 생성한다는 것이다. 중복된 결과가 성능에 미치는 영향은 다음과 같다. 규칙의 적용 결과 생긴 모든 문장은 triples 테이블에 저장하게 되는데, triples 테이블은 중복된 문장이 없어야 한다. 따라서 규칙 적용 후 생긴 결과에 대해 triples 테이블에 이미 존재하는지 체크해야 한다. 따라서 만일 중복된 결과가 많다면 이런 체크 과정에서 많은 시간을 소모하게 되고, 추론 과정은 비효율적이 된다.

이런 중복된 결과가 생기는 경우의 예는 다음과 같다. rdfs2는 도메인 정보를 이용해 주어로 사용된 리소스의 타입을 추론하는 규칙이며 rdfs4a는 주어로 사용된 모든 리소스에 대해 타입을 rdfs:Resource로 정의하는 규칙이다. 이때 프로퍼티의 도메인이 rdfs:Resource로 되어 있는 경우가 있다면 이 정보를 사용한 rdfs2번 규칙의 적용은 rdfs4a를 적용한 결과와 중복된다. 이런 중복된 결과의 발생은 RDFS 함의 규칙 자체가 가진 중복성에 의한 것으로 볼 수 있다. 즉 여러 규칙의 적용이 같은 결과를 만들어 내는 경우가 있기 때문이다[7]. 따라서 중복된 결과에 대한 고려 없이 함의 규칙을 적용하면 많은 비효율이 발생 할 수 있다.

이런 문제점들은 함의 규칙간 의존성이 규칙의 의미에 대한 고려 없이 규칙의 전제와 결론의 패턴만을 보고 정해졌기 때문이다. 규칙간의 의미를 고려하고 그에 따라 규칙 적용의 순서를 정한다면 불필요한 규칙 적용을 줄이고 중복된 결과에 대한 처리를 할 수 있을 것이다.

4.3 규칙 적용 순서를 이용한 전방향 추론 최적화 기법

여기서는 전방향 추론시의 성능 개선을 위한 규칙 적용 순서를 제안하고자 한다. 이를 위해 함의 규칙의 의미에 대해 좀더 살펴보자.

4.3.1 함의 규칙의 분류

13개의 함의 규칙은 규칙이 갖는 의미에 따라 다음과 같이 분류 할 수 있다[7].

- Category-1 리소스에 대한 기본 타입 지정 규칙: rdf1, rdfs4a, rdfs4b
- Category-2 프로퍼티 도메인, 레인지 정보를 이용한 리소스의 타입 추론 규칙: rdfs2, rdfs3
- Category-3 클래스 계층관계에 대한 추론: rdfs8, rdfs10, rdfs11, rdfs13
- Category-4 프로퍼티 계층관계에 대한 추론: rdfs5, rdfs6, rdfs12
- Category-5 클래스 계층관계를 이용한 추론 규칙: rdfs9
- Category-6 프로퍼티 계층관계를 이용한 추론 규칙: rdfs7

결국 RDFS 추론이란 리소스의 타입 추론과 클래스, 프로퍼티의 계층관계 생성과 그 계층관계를 이용한 추론으로 볼 수 있다.

4.3.2 함의 규칙 적용 순서

본 논문에서 제안하는 규칙 적용 순서는 그림 3과 같다.

Sesame의 추론 알고리즘과는 다르게 각 규칙을 적용하며 생성된 결과는 S_{new}에 추가되어 다음 규칙에 적용되고 있음을 알 수 있다. 또한 규칙 적용 순서가 지정되어 있음을 알 수 있다(규칙 적용 순서를 의미하기 위해 <> 기호를 사용하여 규칙 리스트를 표현하였다).

규칙 적용 순서의 의미는 다음과 같다. 우선 RDF 데이타 자체를 순수 RDF 데이타와 스키마 데이타로 나누어 생각하자. 이 스키마 데이타란 클래스와 프로퍼티의 계층관계를 나타낸다. 추론의 첫 단계에서는 새롭게 저

S_{old} = Set of statements already stored
S_{new} = Set of statements newly stored
//Application of Category - 1 rules
for $r \in \langle rdf1, rdfs4a, rdfs4b \rangle$
    $S_{new} = S_{new} \cup applyRule(r, S_{new})$
//Application of Category - 5,6,2 rules
for $r \in \langle rdfs7, rdfs2, rdfs3, rdfs9 \rangle$
    $S_{new} = S_{new} \cup applyRule2(r, S_{new}, S_{new} \cup S_{old})$
//Application of Category - 3 rules
for $r \in \langle rdfs13, rdfs8, rdfs10 \rangle$
    $S_{new} = S_{new} \cup applyRule(r, S_{new})$
$T = S_{new}$
Do until($T = \phi$)
    $T = applyRule2(rdfs11, S_{new} \cup S_{old}, T)$
    $S_{new} = S_{new} \cup T$
//Application of Category - 4 rules
for $r \in \langle rdfs6, rdfs12 \rangle$
    $S_{new} = S_{new} \cup applyRule(r, S_{new})$
$T = S_{new}$
Do until($T = \phi$)
    $T = applyRule2(rdfs5, S_{new} \cup S_{old}, T)$
    $S_{new} = S_{new} \cup T$
$H = \phi$
//Application of Category - 5,6,2 rules
for $r \in \langle rdfs7, rdfs2, rdfs3, rdfs9 \rangle$
    $H = H \cup applyRule2(r, S_{new} \cup S_{old}, S_{new})$
    $S_{new} = S_{new} \cup H$

그림 3 규칙 적용 순서

장된 리소스의 기본 타입을 지정하게 된다(Category-1). 그 후 이미 저장소에 저장된 스키마 데이타를 이용해 리소스들의 타입을 추론한다(Category-5,6,2). 그러나 입력된 데이타에는 순수 RDF 데이타뿐만 아니라 스키마 데이타도 포함될 수 있다. 따라서 입력 데이타에 있는 스키마 데이타를 추출하여 클래스와 프로퍼티의 계층관계를 만드는 과정이 뒤 따른다(Category-3,4). 마지막 단계에서는 이렇게 새로 만들어진 스키마 데이타를 사용하여 입력 데이타뿐만 아니라 이전에 저장되어 있는 데이타에 대해 리소스 타입 추론을 한다(Category-5,6,2).

위의 추론 과정에서 rdfs11과 rdfs5는 더 이상 결과가 나오지 않을 때까지 적용한다. 이는 자기 자신에 의존성이 있는 규칙이기 때문인데, 다음절에서 좀더 자세히 설명하도록 한다. 또한 Category-5,6,2에 해당하는 규칙은 두 번 적용하는 것을 볼 수 있는데, 이 때 $applyRule2(r, S_1, S_2)$의 인자의 순서가 바뀌었음을 주목해야 한다. 규칙 적용이 끝나면 마지막 단계에서 추론한 결과를 갖고 있는 H에 대해 다음 사항을 조사하여 추론을 되풀이 해야 하는지를 결정한다.

- RDFS에서 정의된 기본 클래스에 대한 하위 클래스가 있는가?
- RDFS에서 정의된 기본 프로퍼티에 대한 하위 프로퍼티가 있는가?

만일 위의 경우가 존재한다면 추론 과정을 반복해야 한다. 추론 과정을 반복할 때에는 무조건 처음부터 반복하는 것이 아니라 어떤 하위 클래스, 하위 프로퍼티가 생성되었는지에 따라 반복을 시작하는 위치를 조정할 수 있다. 만일 rdfs:subClassOf의 하위 프로퍼티가 생성되었다면 rdfs11을 적용하는 부분부터 시작하면 된다.

이와 같이 추론이 반복되어야 하는 경우는 RDFS의 클래스, 프로퍼티에 대한 정보가 변경된 경우라고 할 수 있다. 만일 새로 저장된 데이타에서 ns:isA가 rdfs:subClassOf의 하위 프로퍼티로 정의되었다고 하자. 이런 경우 ns:isA는 rdfs:subClassOf와 RDFS 상의 의미론적 관계가 동일하다. 즉 ns:isA 프로퍼티도 추이성과 같은 성질을 갖게 되며 따라서 rdfs:subClassOf에 대해 적용된 규칙들이 ns:isA에 대해서도 적용되어야 한다. 이런 경우 불가피하게 추론 과정을 반복해야 한다. 이러한 반복을 줄이기 위해서는 이와 같이 RDFS와 관련된 정보에 대한 데이타는 초기 단계에서 미리 저장을 해두는 것이 유리하다.

Sesame의 추론 엔진에서는 추론의 대상인 new_triples 테이블이 규칙 적용의 한 반복이 끝난 후에 새로 생긴 문장의 테이블로 교체가 되었다. 그러나 위의 규칙 적용 순서를 사용할 때에는 한 규칙의 적용 후 바로 새로 생긴 문장을 new_triples 테이블에 추가 해야 한다. 이는 다음 규칙의 적용 대상이 이전 규칙의 적용 결과까지 포함해야 하기 때문이다. 이 순서로 규칙을 적용하였을 때 앞에서 언급한 문제들은 다음과 같이 해결된다.

- 추이성 추론에 따른 규칙 적용의 반복: rdfs7_2, rdfs9_2는 규칙 순서에서 클래스, 프로퍼티의 계층관계를 모두 적용한 뒤에 한번씩만 적용하게 된다.
- RDFS의 기본 어휘의 확장: 기본 어휘의 확장이 있는 경우는 모든 규칙의 적용이 끝난 후에 필요에 따라서만 규칙 적용을 반복한다. 따라서 불필요한 규칙 적용을 줄일 수 있다.

이와 같은 규칙 적용 순서를 적용하기 위해서는 다음과 같은 제약이 필요하다. RDFS의 기본 클래스의 상위 클래스와 기본 프로퍼티의 상위 프로퍼티가 존재하지 않아야 한다. RDFS는 기본적으로 데이타를 기술에 대한 제약이 거의 없는 모델이다. 기본 클래스, 기본 프로퍼티에 대해 상위 클래스, 상위 프로퍼티를 정의하는 것이 역시 RDFS의 데이타 모델을 위반하는 것은 아니다. 그러나 이러한 무제약이 오히려 데이타 모델에 대한 모호성을 증가시키고 있기 때문에 [11]에서는 RDFS 모델을 몇 단계의 계층으로 구분할 것을 제안하였다. [11]에서는 RDFS의 기본 어휘와 사용자가 정의할 수 있는 어휘는 서로 다른 계층을 갖고 있으며 기본 어휘에 대

한 확장을 막고 있다. 이러한 모델은 RDFS 모델에 약간의 제약을 추가하여 모호함을 해결해준다. RDFS의 기본 클래스, 기본 프로퍼티에 대해 상위 클래스, 상위 프로퍼티를 지정하는 것은 이러한 제약을 위반하는 것으로 볼 수 있으며 실제로 유용하지 않은 결과를 초래한다. 따라서 우리가 제안한 규칙 적용 순서에서 가정한 제약은 실제 데이타를 다루는 데 있어서 큰 영향을 미치지 않는다고 볼 수 있다.

4.3.3 규칙 적용 순서의 검증

이와 같은 순서로 추론을 하였을 때 완전한 추론 결과를 얻을 수 있는지에 대한 검증이 필요하다. Sesame의 추론 방법은 완전하다는 것은 쉽게 알 수 있다. 기본적인 전방향 추론 과정에서 트리거 되지 않는 규칙의 적용만 제거 하였고 반복적으로 규칙을 적용하는 구조는 바뀌지 않았기 때문이다. Sesame 추론 과정에서 완벽성을 보장하는 중요한 성질은 한 반복에서 결과를 생성한 규칙과 의존성이 있는 규칙을 다음 번 반복에서 적용한다는 것이다.

마찬가지로 우리가 제안한 방법을 검증하기 위해 그림 3에서의 추론 과정을 한 반복으로 보고, 한 반복 후에 필요한 규칙 의존성을 생각해보자. 이는 표 2를 기본으로 하여 불필요한 의존성을 제거 하는 방식으로 알 수 있다. 다음과 같은 방법으로 의존성을 제거한다.

• 가정에 따른 의존성 제거

우리는 앞에서 추론 순서를 적용하기 위해 몇 가지 가정을 하였다. 이 가정이 성립되면 몇 가지 의존성은 고려 하지 않아도 된다. 즉 RDFS 에서 정의된 클래스, 프로퍼티의 상위 클래스, 상위 프로퍼티가 존재하는 경우에만 성립하는 규칙 의존성을 제거한다.

• 규칙 적용 순서에 의한 의존성 제거

각 규칙에 대해 의존성이 있는 규칙 중 그 규칙 보다 나중에 적용되는 규칙을 제거한다. 이런 경우는 추론 과정의 한 반복 후 더 이상 의존성이 남아있지 않게 된다. 이는 이들 의존성의 규칙만을 고려하였을 때, Sesame의 추론 결과와 우리가 제안한 순서에서 한번의 반복으로 추론하였을 때 같은 결과가 됨을 보이면 된다. 이는 다음과 같이 보일 수 있다.

S를 문장의 집합이라고 하고 $rdfs_N(S)$를 문장 S 집합 에 rdfsN 규칙을 적용한 결과 생성된 문장의 집합이라 하자. 표 2의 규칙 중 $rdfsA \rightarrow rdfsB$인 두 규칙 rdfsA, rdfsB에 대해 우리가 제안한 방법에서의 규칙 적용 순서가 rdfsA, rdfsB라고 하자. 초기 문장의 집합 S를 Sesame 알고리즘으로 추론하게 되면 첫 반복에서는 rdfsA, rdfsB를 적용하고 다음 반복에서는 의존성에 의해 rdfsB를 한번 더 적용한다(rdfsA의 적용 결과가 있다고 가정한다). 이때 추론 후 결과는 다음과 같다.

$$rdfsB(rdfsA(S)+rdfsB(S))+rdfsA(S)+rdfsB(S)+S$$
$$=rdfsB\cdot rdfsA(S)+rdfsB\cdot rdfsB(S)+rdfsA(S)+rdfsB(S)+S \quad (1)$$

우리가 제안한 방법에서는 이전 규칙의 적용 결과가 다음 규칙의 적용 대상에 포함된다. $rdfs'_N(S)=rdfs_N(S)+S$라 정의하자. 우리가 제안한 순서대로 적용을 하였을 때 결과는 다음과 같다.

$$rdfs'B(rdfs'A(S))=rdfs'B(rdfsA(S)+S)$$
$$=rdfsB\cdot rdfsA(S)+rdfsB(S)+rdfsA(S)+S \quad (2)$$

(1)과 (2)의 차이는 $rdfsB\cdot rdfsB(S)$이다. 이때 다음이 성립한다.

$$rdfsB \nrightarrow rdfsB \text{ 이면 } rdfsB\cdot rdfsB(S)=\phi$$

즉 $rdfsB \nrightarrow rdfsB$인 경우에는 (1)과 (2)의 결과가 같게 된다. 즉 Sesame의 추론 결과와 우리가 제안한 추론 결과가 같게 된다. 그러나 표 2에서도 알 수 있듯이 자기 자신에게 의존적인 규칙은 여러 개가 존재한다. 우선 rdfs2_1, rdfs3_1이 있다. 그러나 이 경우는 rdf:type의 도메인과 레인지를 이용한 추론을 하게 된다. 그러나 rdf:type의 도메인은 rdfs:Resource, 레인지는 rdfs:Class이기 때문에 추론 결과가 이미 나왔다는 것은 쉽게 알 수 있다. 또한 rdfs7_1, rdfs9_2는 앞에서 가정을 위반하는 의존성으로 제거가 된다. 마지막으로 rdfs5, rdfs11은 추론 과정에서 반복적으로 적용하도록 하였다. 앞에서 말한 이 두 규칙의 적용 방법에 대한 이유가 바로 이 재귀적 의존성 때문이었다. 결국 추론 과정에서 결과가 더 이상 나오지 않을 때까지 적용하기 때문에 이 의존성도 삭제 할 수 있다.

위와 같이 의존성을 제거하면 표 3과 같은 의존성만 남게 된다. 남은 의존성들은 앞에서 말한 추론을 반복해야 하는 경우에 해당하며 추론을 반복하게 되면 의존성을 만족 시킨다. 따라서 우리가 제안한 추론 순서가 추론 결과의 완전성을 보장함을 알 수 있다.

4.4 중복된 결과의 제거

고갈적 전방향 추론을 사용하면 추론 과정에서 많은 중복된 결과가 생길 수 있다. 이는 앞에서 언급한대로 RDFS 함의 규칙이 중복성을 갖고 있기 때문이다. 중복된 결과가 성능에 미치는 영향을 알아보기 위해 다음의 SQL을 보자.

이 SQL은 rdfs11을 적용한 결과를 만든다. 여기서 진한 글씨로 표시된 부분은 규칙 적용 결과가 triples 테이블에 없어야 하는 조건을 나타낸다. 즉 SQL의 중간 결과에 대해 triples 테이블에서의 존재 유무를 체크하여 존재하지 않는 결과만 반환한다. 이 과정이 모든 중간 결과에 대해 행해져야 하기 때문에 중간 결과의 크기를 될 수 있는 한 줄여야 한다. 중간 결과의 많은

표 3 규칙 순서 변경에 따른 함의 규칙간 의존성 테이블

규칙	1	2_1	2_2	3_1	3_2	4a	4b	5_1	5_2	6	7_1	7_2	8	9_1	9_2	10	11_1	11_2	12	13
1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2_1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
2_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3_1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
3_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4a	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
4b	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5_1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5_2	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7_1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
7_2	-	-	-	-	-	-	-	X	X	X	X	X	X	X	-	X	X	X	X	X
8	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9_1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
9_2	-	-	-	-	-	-	-	-	-	X	-	-	X	-	-	X	-	-	X	X
10	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-
11_1	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-
11_2	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-
12	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-
13	-	-	-	-	-	-	-	-	-	-	X	-	-	-	-	-	-	-	-	-

부분이 이미 triples 테이블에 존재한다면 추론 성능의 저하로 나타난다.

우리가 제안하는 방법에서는 규칙 적용 순서를 정했기 때문에 앞에서 적용한 규칙에 의해 어떠한 패턴의 문장이 생성되었는지 예측 할 수 있다. 다음은 각 규칙의 적용에서 중복된 결과의 생성을 줄이기 위해 적용한 방법이다. 각 규칙을 적용하는 SQL에 WHERE 조건을 추가하여 triples에 이미 있는 중간 결과를 최대한 줄이는 효과를 얻을 수 있다.

- rdfs2(rdfs3): 도메인(레인지)가 rdfs:Resource인 프로퍼티는 고려 대상에서 제외한다. 이는 앞서 적용한 rdfs4a(rdfs4b)에서 모든 리소스에 대해 타입을 rdfs:Resource로 추론하였기 때문이다.
- rdfs7(rdfs9): 프로퍼티 rdfs:subProperyOf(rdfs:subClassOf)의 반사성(reflexive property)으로 인한 중복된 결과 생성을 제거한다.
- rdfs5, rdfs11: 이 두 규칙은 rdfs:subProperyOf, rdfs:subClassOf에 대한 추이성에 대해 추론하는 규칙이다. 이 규칙을 적용하는 것에 대해서는 다음 추이성 추론에서 좀더 자세한 설명을 하겠다.

4.4.1 추이성 추론

추이성 추론은 클래스, 프로퍼티의 계층관계를 생성하기 위해 사용된다. 계층관계가 깊은 데이타를 저장할 때에는 추이성 추론 과정에서 중복된 결과가 많이 생기며 많은 부하를 가져온다. rdfs11을 대상으로 추이성 추론에서의 중복 결과 제거 방안을 설명하겠다. rdfs5도 프로퍼티만 다를 뿐 같은 방법으로 적용된다. 우리가 제시한 추론 순서에 의하면 이 rdfs11은 더 이상 결과가 생성되지 않을 때까지 반복하여 적용한다. 이는 추이성 닫힌 집합을 계산하기 위함이다. 이때 어떤 중복된 결과가 생기게 되는지 알아보자.

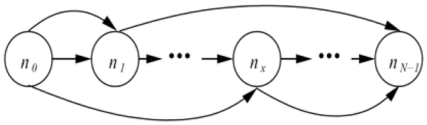


그림 4 노드 경로 그래프

그림 4와 같이 N개의 노드 $n_x, 0 \le x \le N-1$와 n_y, n_{y+1} 사이의 간선 $e_y, 0 \le y \le N-2$가 있다고 하자. 각 노드는 클래스를 나타내며 간선은 rdfs:subClassOf 관계를 나타낸다. 초기에는 노드 n_y, n_{y+1}사이의 간선만 존재한다고 가정한다. 이런 상황에서 rdfs11을 적용하면 점차 긴 길이를 갖는 간선이 생기고 결국 N-1 길이의 간선까지 생길 것이다. 이때, N-1 길이의 간선이 생기는 상황을 생각해보자. 이미 N-1 보다 짧은 간선이 모두 생성되었고, N-1길이의 간선은 길이가 1인 간선과 길이가 N-2인 간선의 결합으로 생긴다. 따라서 N-1 길이의 간선이 생기게 되는 경우의 수는 두 노드 사이의 노드 개수인 N-2가 된다. 결국 마지막 규칙 적용 시에 N-2개의 중복된 결과가 생기게 됨을 알 수 있다. 따라서 N-2개의 같은 결과에 대해 중복 체크를 해야 하는 비효율

이 발생한다. 이런 점을 개선하기 위해 다음과 같이 SQL을 변경하였다. 즉 생성된 결과에 대해 중복 체크를 하기 전에 유일한 값만을 선별하는 과정을 거쳐서 중복된 결과에 대한 체크를 줄인 것이다.

```
SELECT t.subject, <rdfs:subClassOf>, nt.object
FROM triples t, newtriples nt
WHERE t.predicate = <rdfs:subClassOf> AND t.object = nt.subject AND
        nt.predicate = <rdfs:subClassOf> AND
        (t.subject, <rdfs:subClassOf>, nt.object)
        NOT IN (SELECT subject, predicate, object FROM triples)
```

5. 실험 결과

구현은 Sesame 1.1.3을 사용하였으며 Java SDK 1.5.0으로 구현하였다. RDBMS로는 MySQL 4.1.2를 사용하였다. 실험 환경은 Pentium M 730(1.6GHz), 1GB 램을 갖는 사양의 시스템을 사용하였으며 운영체제로는 Windows XP Professional을 사용하였다. 실험은 벌크 데이타 저장시의 추론 성능과 추론 방식의 변화에 따른 데이타 저장 성능의 확장성의 변화 대해 측정하였다. 벌크 데이타 저장은 큰 용량을 갖는 데이타 파일을 한번에 저장하는 상황에 대해 실험을 한 것이라면 확장성에 대한 실험은 데이타 업데이트 시에 발생하는 상황에 대한 실험이다.

5.1 벌크 데이타 저장

실험에 사용한 데이타는 Wordnet[3], NCI Cancer Ontology(NCI), Gene Ontology(GO)[4]이다. 이 데이타들은 비교적 많은 문장을 갖고 있는 대용량 RDF 데이타들이다. 또한 NCI와 GO 데이타는 깊은 클래스 계층 관계를 갖고 있다. 표 4는 각 데이타에 대한 정보와 추론된 결과를 나타낸다.

표 4에서 Sesame의 방법과 우리가 제안한 방법이 동일한 개수의 추론된 문장을 생성한 것을 알 수 있다. 이는 우리가 제안한 방법이 완벽한 추론 결과를 생성함을 보여주는 것이다. 표 5는 추론 성능에 대한 비교를 보여준다. 표 5에서 규칙 적용 횟수란 규칙의 종류에 관계없이 추론 과정 동안 적용된 규칙의 개수를 의미하며 추론 시간은 데이타 저장 과정에서 데이타 파일의 파싱이 끝나 문장 생성이 완료된 직후부터 모든 추론 결과가 생성된 시점까지의 시간을 말한다.

위의 결과는 각각의 데이타에 대해 10개의 측정값을 구한 뒤 평균을 취한 값이다. 결과에서 규칙 적용 횟수가 많이 감소하였음을 알 수 있으며 전체적 추론 성능이 향상하였음을 알 수 있다.

실험 결과를 자세히 분석하기 위해 실험 데이타의 특성을 알아보자. 표 6은 각 실험 데이타에 대해 추론 결과 문장의 비율을 규칙 종류 별로 보여준다. Wordnet과 GO에서 규칙의 적용은 주로 rdfs4a, 4b인 것을 알 수 있다. 그러나 rdfs4a, 4b 규칙은 Category-1에 속하는 규칙으로 리소스의 기본 타입을 지정하는 규칙이다. 이 규칙은 사실 RDF에서 크게 의미를 갖는 규칙으로 볼 수 없다. 다시 말해 Wordnet과 GO 데이타는 그리 많은 RDF semantics를 갖고 있지 않은 데이타라고 할 수 있다. 반면 NCI의 경우에는 rdfs11번의 적용 결과가 가장 많았다. 이는 NCI 데이타 안에 많은 클래스 계층 관계를 갖고 있기 때문이다. 또한 다른 데이타에 비해 rdfs4a, 4b의 비율이 작고 다른 규칙의 비율이 크다는 것을 알 수 있다. 따라서 상대적으로 NCI 데이타는

표 4 실험 데이타

	크기(MB)	문장개수	추론된 문장 개수	
			Sesame	Our approach
Wordnet	48.7	473,626	99,690	99,690
NCI	57.4	851,373	966,827	966,827
GO	275	6,653,592	2,055,383	2,055,383

3) http://www.semanticweb.org/library/
4) http://www.geneontology.org/

표 6 규칙 별 결과 문장의 비율

	Wordnet(%)	NCI(%)	GO(%)
rdf1	0.004	0.005	0.001
rdfs2	0	4.305	0.856
rdfs3	0.006	4.857	0.142
rdfs4a,4b	99.969	9.167	91.593
rdfs5	0	0	0
rdfs6	0.009	0.015	0.001
rdfs7	0	0	0
rdfs8	0.006	9.158	0.998
rdfs9	0	0	0
rdfs10	0.006	9.153	0.998
rdfs11	0	63.344	5.410
rdfs12	0	0	0
rdfs13	0	0	0

표 5 실험 결과

	규칙적용횟수			추론시간		
	Sesame	Our approach	감소비율(%)	Sesame(s)	Our approach(s)	감소비율(%)
Wordnet	46	18	60.9	99.625	79.437	20.3
NCI	53	23	56.6	1108.625	708.219	36.1
GO	53	22	58.5	2973.219	2330.500	21.6

Wordnet과 GO보다 많은 RDF semantics를 갖고 있음을 알 수 있다. 실험 결과에서 추론 시간의 감소 비율이 NCI 데이타가 가장 많다는 것은 이런 점에서 의미를 갖는다고 할 수 있다. 즉 많은 RDF semantics를 갖고 있는 데이타 일수록 추론 과정이 복잡해지며 우리가 제기 했던 추론 상의 문제점이 많이 나타나 성능 향상이 더 많이 된다는 것을 알 수 있다. 앞으로 더욱 많은 RDF semantics을 갖는 데이타가 쓰이고 복잡한 추론을 요하는 상황이라면 성능향상은 더욱 클 것으로 기대된다.

5.2 데이타 저장 성능의 확장성

다음으로는 데이타 저장 성능의 확장성에 대한 측정을 하였다. Sesame는 이미 저장되어 있는 문장의 개수에 따라 저장 성능의 차이를 보인다. 이는 이미 저장되어 있는 문장이 많게 되면 추론 과정에 소요되는 시간이 늘어나기 때문이다. 확장성의 측정은 다음과 같이 실험 하였다. GO 데이타는 생물학 용어를 정의한 온톨로지로 각 용어를 정의한 클래스들의 모음이다. 이 데이타의 각 용어에 해당하는 부분을 각각의 RDF 파일로 만들어 저장하였다. GO에 있는 용어들은 18,577개였고 따라서 18,577개의 RDF 파일을 생성하였다. 이들은 대략 200 byte~15Kbyte의 크기를 가지며 대부분 10개 미만의 문장을 생성하게 된다. 이 파일들을 각각 저장하며 걸린 시간을 측정하였고 1,000개의 데이타로 나누어 저장 시간의 평균을 구하였다. 그림 5는 이 결과로 우리가 제안한 방법을 사용하였을 때 확장성이 향상하였음을 보여준다.

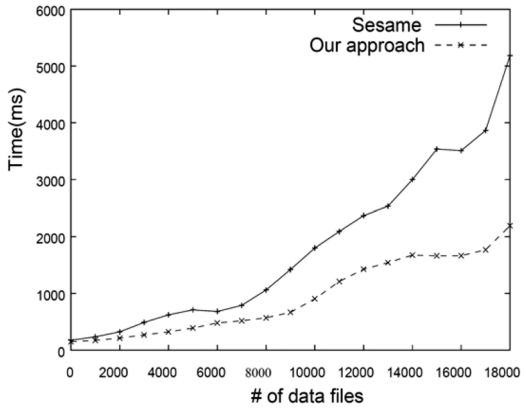


그림 5 확장성 측정

6. 결론과 향후 연구

본 논문에서는 RDFS 추론의 성능을 향상하기 위한 전방향 추론의 최적화 방안을 제안하였다. 전방향 추론시 규칙을 순차적으로 반복 적용할 경우 발생하는 비효율을 설명하였고 이를 해결하기 위한 규칙 적용 순서를 제안하였다. 제안한 추론 방법은 이전의 추론 방법에 비해 좀더 적은 규칙 적용으로도 완벽한 결과를 얻을 수 있음을 보였다. 또한 제안한 규칙 적용 순서를 사용할 때에는 중복된 결과를 생성하는 경우를 줄일 수 있었다.

제안한 방법의 성능을 평가하기 위해 실제 사용되는 데이타들에 대해 추론 성능을 테스트 하였다. 대용량 데이타를 벌크로 저장하는 경우 기존의 방법보다 성능이 20~30% 향상하는 것을 볼 수 있었다. 현재 나와 있는 데이타로 실험을 하였지만 좀더 복잡한 추론 과정이 필요한 데이타에 대한 실험을 한다면 성능향상 폭이 클 것으로 기대된다. 또한 점진적으로 데이타를 업데이트 때의 저장 속도 변화에 있어서도 기존의 방법보다 데이타 저장의 확장성이 향상됨을 볼 수 있었다.

본 논문에서는 RDFS 추론에 대해 다루었지만 OWL에 대한 추론은 규칙이 더욱 다양하고 복잡하다. 따라서 향후 연구로 OWL 추론 엔진의 성능의 최적화에 대한 연구가 필요할 것으로 보인다. 또한 본 논문에서 고려하지 않은 확장 함의 규칙과 데이타 타입 함의 규칙을 이용한 추론에 관한 연구도 필요할 것이다.

참 고 문 헌

[1] Graham Klyne, Jeremy J. Carroll, and Bran McBride. Resource Description Framework (RDF): Concepts and Abstraction. W3C Recommendation, 2004.

[2] Dan Brickley, R.V. Guha, and Brian McBride. RDF Vocabulary Description Language 1.0: RDF Schema. W3C Recommendation, 2004.

[3] Patric Hayes and Brian McBride. RDF Semantics. Technical report, W3C Recommendation, 2004.

[4] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Proceedings of the International Semantic Web Conference, 2002.

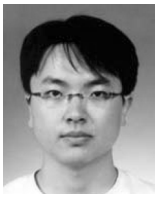
[5] Raphael Volz, Steffen Staab, and Boris Motik. Incremental Maintenance of Materialized Ontologies. CoopIS/DOA/ODBASE, 2003.

[6] Jeen Broekstra and Arjohn Kampman. Inferencing and Truth Maintenance in RDF Schema. In Proceedings of the Workshop on Practical and Scalable Semantic Systems, 2003.

[7] Ora Lassila. Taking the RDF Model Theory Out For a Spin. In Proceedings of the International Sematic Web Conference, 2002.

[8] Yuanbo Guo, Zhengxiang Pan, and Jeff Heflin. An Evaluation of Knowledge Base Systems for Large OWL Datasets. In Proceedings of the International

Semantic Web Conference, 2004.
[9] Jeremy J. Carroll, Ian Dickinson, Chris Dollin, Dave Reynolds, Andy Seaborne, and Kevin-Wilkinson. Jena: Implementing the Semantic Web Recommendations. In Proceedings of the International World Wide Web Conference, 2004.
[10] Charles Forgy. Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem. Artificial Intelligence, 19(1), 1982.
[11] Jeff Z. Pan and Ian Horrocks. RDFS(FA) and RDF MT: Two Semantics for RDFS. In Proceedings of the International Semantic Web Conference, 2003.



김 기 성
2003년 서울대학교 응용화학부(학사)
2003년~현재 서울대학교 전기, 컴퓨터공학부 대학원 박사과정. 관심분야는 데이타베이스, 시맨틱웹, 온톨로지, 생물정보학

유 상 원
정보과학회논문지 : 데이타베이스
제 33 권 제 1 호 참조

이 태 휘
2004년 서울대학교 컴퓨터공학과(학사)
2004년~현재 서울대학교 전기, 컴퓨터공학부 대학원 박사과정. 관심분야는 데이타베이스, 시멘틱웹, 데이타마이닝

김 형 주
정보과학회논문지 : 데이타베이스
제 33 권 제 1 호 참조