RDF 접근 제어에서 소수 그래프 레이블링을 사용한 효율적 권한 충돌 발견

(Efficient Authorization Conflict Detection Using Prime Number Graph Labeling in RDF Access Control)

김 재 훈 † 박 석 ††
(Jaehoon Kim) (Seog Park)

요 약 RDF와 OWL은 시맨틱 웹을 위한 두 가지 핵심 기반 기술이다. 이러한 RDF와 OWL을 이용하는, 또한 이에 관련된 많은 연구들이 최근 소개되었다. 하지만, RDF와 OWL에 대한 정보 보안 관련 연구는 미비한 실정이다. 본 논문에서는 RDF 보안 기술과 관련하여, RDF 트리플에 기반을 둔 안전한 접근 제어 명세 모델을 간단히 소개한다. 다음으로 RDF 접근 제어 명세 시의 추론에 의한 권한 충돌을 효율적으로 발견하기 위하여 소수 그래프 레이블링을 기법을 활용하는 방법을 자세히 소개한다. 추론에 의한 접근 권한 충돌 문제는 비록 하위 개념에 대한 접근 권한이 허용이지만, 하위 개념은 상위 개념으로 추론될 수 있으므로, 만약 상위 개념에 대한 접근 권한이 불허로 되어 있는 경우 하위 개념 또한 허용되어서는 안 되는 문제이다. 몇 가지 실험에서는 제안하는 소수 그래프 레이블링을 사용하는 방법이 기존의 단순한 권한 충돌 발견 방법보다 현저히 나은 성능을 가짐을 보여 준다.

키워드 : RDF, 접근 제어, 권한 충돌, 그래프 레이블링, 소수

Abstract RDF and OWL are the primary base technologies for implementing Semantic Web. Recently, many researches related with them, or applying them into the other application domains, have been introduced. However, relatively little work has been done for securing the RDF and OWL data. In this article, we briefly introduce an RDF triple based model for specifying RDF access authorization related with RDF security. Next, to efficiently find the authorization conflict by RDF inference, we introduce a method using prime number graph labeling in detail. The problem of authorization conflict by RDF inference is that although the lower concept is permitted to be accessed, it can be inaccessible due to the disapproval for the upper concept. Because by the RDF inference, the lower concept can be interpreted into the upper concept. Some experimental results show that the proposed method using the prime number graph labeling has better performance than the existing simple method for the detection of the authorization conflict.

Key words : RDF, Access Control, Authorization Conflict, Graph Labeling, Prime Number

· 본 연구는 한국과학재단 특정기초연구(R01-2006-000-10609-0) 지원으로 수행되었음

† 정 회 원 : 서강대학교 컴퓨터학과 교수
jhkimygk@gmail.com
†† 종신회원 : 서강대학교 컴퓨터학과 교수
spark@dblab.sogang.ac.kr

논문접수 : 2007년 11월 7일
심사완료 : 2008년 1월 8일

정보과학회논문지 : 데이타베이스 제35권 제2호(2008.4)

1. 서 론

웹 사용의 기본 형태는 사용자가 검색 엔진의 도움을 받아 자신이 원하는 웹 문서를 직접 찾고, 웹 문서의 정보를 사용자가 이해하고 이를 활용하는 것이다. 이러한 사용자의 웹 검색의 수고를 덜기 위하여, 과거로부터 지능적인 웹 검색을 위한 다양한 연구와 서비스의 개발이 이루어졌다. 이러한 노력은 결국 현재의 시맨틱 웹(semantic web)[1]이라는 궁극적인 해결 방안을 제시하게끔 하였다. 즉, 웹 문서들이 정보 표현을 위해 지식 추론 시스템에서의 온톨로지와 같은 어떤 개념을 표현하는 공통의 어휘(vocabulary)를 사용함으로써, 이를 읽

는 에이전트로 하여금 자동으로 웹 문서를 이해하고 활용하도록 하는 것이다. RDF(Resource Description Framework)[2]와 OWL(Web Ontology Language)[3]은 이러한 공통의 어휘 표현을 위해 W3C에서 승인한 시맨틱 웹을 위한 두 가지 핵심 기반 기술이다.

최근 Jain과 Farkas는 참고문헌[4]의 연구에서 RDF 데이타 추론에서의 RDF 트리플(triple) 기반의 접근 제어 모델을 소개하였다. 그들의 연구는, 기존 RDF 접근 제어에 관한 몇몇 연구들[5-8]과 달리, RDF 모델의 기본적인 트리플 구조로 보안 객체(security object)를 표현하고, 추론에 의한 접근 권한 충돌(conflict)을 다루고 있기 때문에 큰 의미를 가진다고 생각한다. 추론에 의한 접근 권한 충돌의 문제는 간단히 설명하면 다음과 같다. 그림 2의 RDF 웹 문서 예에서 Dave라는 사용자에게 MedicalSubject 클래스에 관련된 정보를 보는 것이 허용되어 있다고 가정하자. 그러면 Dave는 MedicalSubject 클래스의 인스턴스인 Pinocchio에 관련된 정보를 브라우징할 수 있을 것이다. 하지만 그림 1의 상/하위 개념 관계를 나타내는 RDF 그래프에서 Dave에게 Outpatient 클래스에 관련된 정보를 보는 것이 허용되어 있지 않다고 가정하자. 이럴 경우 MedicalSubject는 Outpatient의 하위 클래스(subClassOf 관계)로 포함 관계 기반 추론(subsumption relationship based inference)에 의하여 Pinocchio는 또한 Outpatient로 해석될 수 있다. 따라서 추론에 의한 권한 충돌에 의하여 Dave에게는 Pinocchio에 관한 정보를 보여 주지 말아야 한다.

Jain과 Farkas는 이러한 추론에 의한 권한 충돌의 발견을 위하여 매우 단순 소모적인(brute force) 알고리즘을 소개한다. 그들의 알고리즘은 먼저 추론에 의해 모든 RDF 트리플에 대하여 권한 전파(authorization propagation)를 수행한다. 다음으로 모든 RDF 트리플에 대하여 권한 충돌이 있었는지를 조사하고, 만약 있었다면 그 추론은 취소된다. 이러한 방법은 RDF 데이타가 작은 규모가 아닐 때 비효율적이다. 따라서 본 연구에서는 추론시의 효율적인 권한 충돌의 발견을 위하여 소수(prime number) 그래프 레이블링 기법[9]을 접근 권한 명세에 이용하는 방법을 소개한다. 제안하는 방법은 모든 RDF 트리플을 조사하는 것이 아니라, 포함 관계 기반 추론에 근거하여 현재 명세된 접근 권한과 상/하위 관계를 갖는 클래스들의 해당 접근 권한만을 조사함으로써, 탐색 범위를 현저히 줄이는 것이다. 또한 권한 명세와 권한 추론 사이에서의 포함 관계에 따라 발생할 수 있는 충돌 유형을 분석함으로써, 모든 충돌 유형을 조사하지 않고, 상위 클래스(class) 혹은 속성(property)이 접근 불허(-)의 권한을 갖고 하위 클래스 혹은 속성이 접근 허용(+)의 권한을 갖는 경우만을 조사한다. 몇 가지 실험을 통하여, RDF 데이타의 규모가 커질수록, 또한 명세되는 접근 권한의 수가 증가할수록, 제안하는 기법의 권한 충돌 발견 시간이 단순 소모적인 방법에 비교하여 현저히 나음을 확인하였다.

본 논문의 구성은 다음과 같다. 먼저 2장에서는 RDF 접근 제어 보안 기술과 관련한 현재의 몇몇 연구들 및 그래프 레이블링에 관한 연구들을 소개한다. 3장에서는 RDF 모델과 RDFS(RDF Schema)에 대하여 간략히 소개하며, 4장에서는 RDF 추론에 의한 접근 권한 충돌 문제를 자세히 분석한다. 5장에서는 분석 결과를 바탕으로 제안하는 소수 그래프 레이블링 기법을 이용하여 효율적 권한 충돌 발견을 수행하는 알고리즘을 소개한다. 6장에서는 제안하는 방법의 보다 향상된 충돌 발견 수행 성능을 보여 주는 몇 가지 실험 결과를 제시하며, 7장에서는 본 연구의 결론을 맺는다.

2. 관련 연구

본 절에서는 먼저 RDF 접근 제어에 관한 몇몇 관련 연구에 대하여 소개하며, 다음으로 그래프 레이블링 기법에 관한 연구를 소개한다. 또한 본 연구가 이러한 연구와 어떤 차별성을 갖는 지를 간단히 언급한다.

• RDF 접근 제어에 관한 연구

RDF/RDFS는 XML(eXtensible Markup Language)로 표현되므로, 기존의 연구되어진 XML 접근 제어를 RDF에 적용하는 것을 고려할 수 있다. Damiani et al[10]과 E. Bertino et al.[11,12]는 XML 문서에 대한 세밀한(fine-grained) 접근 제어 모델을 제시하였다. 명세된 접근 제어에 따라, XML 트리 구조에서의 각 노드를 사용자에게 보이지 않도록(invisible)한다. 하지만 서론에서 언급한 것처럼, RDF 접근 제어 모델은 추론에 의해 새롭게 생성된 정보에 대한 접근 제어를 고려할 수 있어야 한다. 기 개발된 XML 접근 제어 모델들은 이러한 문제를 고려하지 않는다.

Jain과 Farkas[4]는 추론을 고려한 RDF 트리플에 기반한 새로운 RDF 접근 제어 모델을 제시하였다. 하지만 Jain과 Farkas는 그들의 연구에서 RDF 트리플 기반의 접근 권한을 보안 관리자가 어떻게 명세하며, 명세의 의미가 어떻게 해석될 수 있는 지를 설명하고 있지 않다. 본 논문에서는 본 연구에서 정의한 RDF 접근 권한의 명세 방법에 대하여 설명하며, RDF 접근 권한 명세시의 추론에 의한 권한 충돌의 문제를 자세히 분석한다. 다음으로 Jain과 Farkas의 모든 RDF 인스턴스에 대해 접근 권한을 할당한 후 충돌 여부를 조사하는 매우 단순 소모적인(brute force) 방법에 반하여, 소수 그래프 레이블링 기법을 이용한 효율적 권한 충돌 발견 방법을 제시한다.

Qin과 Atluri[5], 그리고 Javanmardi et al.[6]은 온톨로지 개념들 사이의 다양한 관계에 있어서의 추론에 의한 권한 전파 문제 및 충돌 문제를 소개한다. 즉, RDF의 포함 관계에 관련된 subClassOf, subPropertyOf 외에 두 개념 사이의 동등 관계(equivalence relationship), 전체 개념과 부분 개념 사이의 관계(예로, intersection, union), 비 추론적 관계(non-inferable relationship)등 다양한 의미적 관계에 대한 권한 전파를 고려한다. 하지만 그들의 연구는 단지 일반적인 개념적 소개의 수준이며, 현재 웹의 온톨로지 표준 언어인 RDF와 OWL 모델의 구조 및 의미를 고려하여, 더욱 구체적인 연구가 필요할 것으로 사료된다. 본 논문에서 제안하는 RDF 접근 제어 모델을 OWL 데이타에 적용할 수 있도록 확장하는 것은 본 연구의 향후 연구 과제이다.

• 그래프 레이블링에 관한 연구

Christophides et al.[13]은 XML 트리(tree) 레이블링으로 연구되어진 몇몇 방법들[14-16]을 그래프 레이블링으로 확장한다. RDF와 OWL등은 방향성 비순환 그래프 DAG(Directed Acyclic Graph)로 표현될 수 있으며, 그래프 레이블링을 통한 효율적인 RDF 질의 처리 수행을 소개한다. Christophides et al.은 Agrawal et al.[14], Li와 Moon[15]의 일종의 인터벌(Interval) 방법, 그리고 Dewey 기법[16] 등을 그래프 레이블링으로 확장하는 방법을 소개한다.

Wu et al.[9]은 Christophides et al.의 확장 방법들이 그래프의 두 노드 사이의 조상/후손 관계를 알기 위하여 많은 반복적인 조인(Join) 연산을 요구하므로, 좀 더 효율적인 방법으로 소수(prime number)를 이용한 그래프 레이블링 방법을 소개한다. 소수를 이용한 방법에서는 조상/후손 관계를 알기 위하여 단 한 번의 나머지 연산(modulo)만이 요구된다. 소수를 사용하는데 있어서의 기본적인 문제점은 소수가 커질수록 나머지 연산 및 곱셈 연산의 수행 시간이 커지는 것인데, Wu et al.은 그러한 연산 시간을 줄일 수 있도록 몇 가지 최적화 방법들을 추가적으로 제안한다. 본 논문에서는 이러한 소수 그래프 레이블링 기법을 접근 제어 충돌 발견에 어떻게 적용할 수 있는 지를 자세히 소개하도록 한다.

3. RDF/RDFS

RDF는 웹 사용자들의 웹 문서에 기술된 정보의 단순한 브라우징을 떠나서, 응용프로그램 혹은 에이전트들이 그러한 정보를 자동으로 해석할 수 있도록 하기 위하여 개발되었다. 이것은 온톨로지와 같이 데이타 의미의 상호 해석을 위한 표준 어휘(vocabulary) 및 그러한 어휘의 관계를 정의할 수 있는 어떤 메커니즘을 제공하기 때문에 가능하다. RDFS(RDF Schema)는 클래스(class) 및 속성(property)의 정의 그리고 그들 간의 관계(subClassOf와 subPropertyOf)를 정의하며, RDF 데이타는 그러한 클래스 및 속성에 해당하는 인스턴스에 관하여 기술한 것이다. 현재 W3C에서 승인한 웹의 표준 온톨로지 언어는 OWL인데, OWL은 RDF 모델을 기반으로 하고 있으며, RDF에서의 클래스와 클래스, 속성과 속성 간의 관계를 정의하는 어휘를 더욱 확장한다. 예로, OWL은 equivalentClass, equivalentProperty, sameAs, unionOf, complementOf, intersectionOf등과 같은 다양한 어휘를 갖는다. RDF는 소규모의 온톨로지 언어로 사용될 수 있으며, 또한 W3C의 RDF semantics[17]에서는 RDF에서 고려될 수 있는 추론에 대한 가이드라인을 제시하고 있다.

그림 1은 하나의 예로 병원과 관련된 RDF와 RDFS를 그래프로 표현한 것이다. RDFS 그래프에서 IsolatedPatient 클래스는 Inpatient 클래스의 하위 클래스(subClassOf)이고, Inpatient 클래스는 Patient 클래스의 하위 클래스이다. 또한 hasMedicalDisease는 hasGeneralDisease의 하위 속성(subPropertyOf)이고 hasGeneralDisease는 hasDisease의 하위 속성이다. 그림 1에서의 RDF 그래프는 그림 2의 RDF 인스턴스 데이타를 표현한 것이다. 그림 2의 어떤 병원 웹 사이트의 RDF 웹 문서에는 Pinocchio가 감기, Shrek이 장염에 걸렸다는 정보가 들어가 있다.

4. RDF 접근 명세에서의 추론에 의한 권한 충돌 문제

4.1 접근 권한 명세

권한 충돌 문제를 살펴보기에 앞서, 본 연구에서 정의하고 있는 RDF 접근 권한 명세 모델을 소개한다. 본 연구의 접근 권한 명세는 RDF의 트리플 구조를 따르기 때문에, 보안 관리자로 하여금 접근 권한 명세를 쉽고 명료하게 수행할 수 있도록 한다.

정의 1 (보안 객체 = RDF 패턴)

RDF 패턴(pattern)은 RDF 트리플(triple) 구조의 $[r, p, v]$ 형태를 가지며, RDF/RDFS 그래프에서의 임의의 트리플과 매칭된다. 여기서 r, p는 임의의 변수 $\$x$, $\$y$ 등으로 치환될 수 있으며 v는 항상 임의의 변수 $\$z$이다. 그리고 $r \in R$, $p \in P$, $v \in V$이다.

- 집합 R은 RDF/RDFS에서의 클래스(혹은 개념, concept) 또는 인스턴스를 가리키는 URI(Uniform Resource Identifier) 노드들과 공백 노드(blank node)들을 포함한다.
- 집합 P는 클래스의 속성(property)을 가리키는 URI들이다.
- 집합 V는 속성에 의해 관계되어지는 다른 클래스 혹

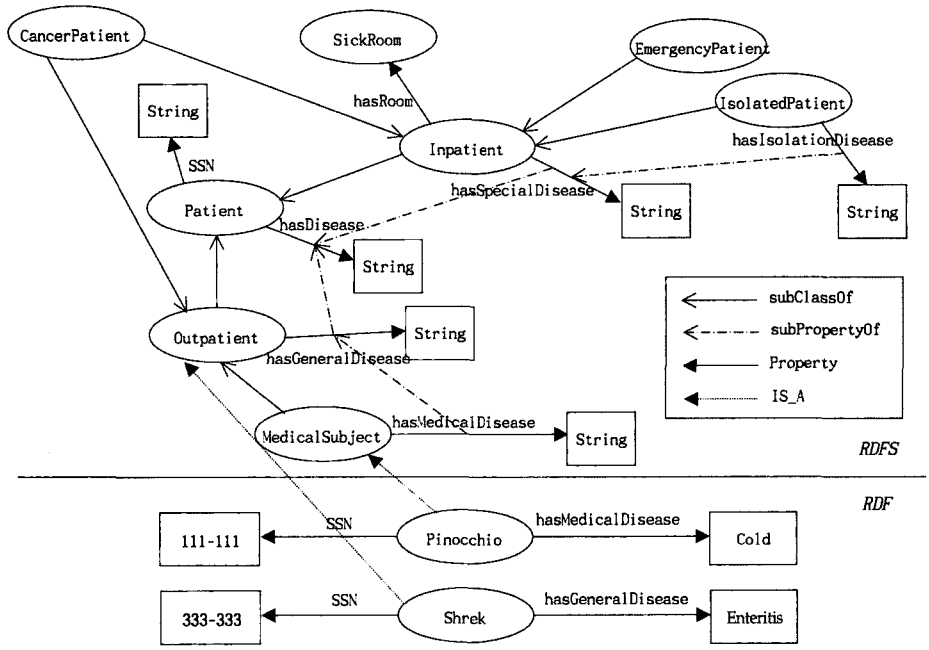


그림 1 병원 관련 RDF/RDFS 그래프의 예

```
<?xml version="1.0"?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:ex="http://example.org/schemas/weapons#">
  <ex:MedicalSubject rdf:ID="Pinocchio">
    <ex:SSN>111-111</ex:SSN>
    <ex:hasMedicalDisease>Cold</ex:hasMedicalDisease>
  </ex:MedicalSubject>
  <ex:Outpatient rdf:ID="Shrek">
    <ex:SSN>222-222</ex:SSN>
    <ex:hasGeneralDisease>Enteritis</ex:hasGeneralDisease>
  </ex:Outpatient>
</rdf:RDF>
```

그림 2 Pinocchio가 감기, Shrek이 장염에 걸렸다는 정보를 담고 있는 RDF 웹 문서

은 인스턴스의 URI, 공백 노드, 그리고 리터럴(literal) 값을 포함한다.

예로, RDF 패턴 [$x, hasRoom, $z]는 그림 1의 RDF/RDFS 그래프 표현에서의 [Inpatient, hasRoom, SickRoom]와 매칭된다. 또한 매칭 트리플 μ([Pinocchio, SSN, $z]) = {[Pinocchio, SSN, 111-111]}이며, 매칭 트리플 μ([$x, $y, $z])는 그래프의 모든 에지를 매치한다.

정의 2 (RDF 접근 권한).

RDF 접근권한 a_i는 <*subject*, *object*, *action*, *sign*, *type*>의 다섯 개의 튜플로 구성된다.

- *subject*는 권한이 주어지는 주체이다.
- *object*는 권한이 적용되는 객체로 정의 1의 RDF 패턴에 의하여 매칭되는 RDF 트리플들이다.
- *action*은 *object*에 대해 수행되는 연산을 기술하며, 본 연구에서는 웹 문서 상에서의 RDF 데이타의 접근 제어를 고려하므로, 읽기 연산으로 제한한다.
- *sign* ∈ {+, -}이며, + 부호에 의해 *action*이 허용됨을 - 부호에 의해 *action*이 거부됨을 표시한다.
- *type* ∈ {L, R}이며, L(Local)은 권한의 전파(propagation)가 해당 RDF 트리플에만 적용되는 것을 표시하며, R(Recursive)는 권한의 전파가 해당 RDF 트리플의 모든 하위 RDF 트리플(이것은 subClassOf, subPropertyOf의 포함 관계에 의하여 정의되어짐)에 적용됨을 표시한다.

4.2 추론에 의한 권한 충돌 문제

본 절에서는 접근 권한이 명세 될 경우, RDF 추론에 의한 권한 충돌 문제를 살펴본다. 본 연구에서는 RDF 추론의 핵심이 되는 포함 관계(subsumption) 기반 추론, 즉 subClassOf와 subPropertyOf에 초점을 맞춘다. subClassOf 추론은 어떤 클래스 c_i가 c_j의 하위 클래스일 경우($c_i \subset c_j$), c_i가 c_j로 해석될 수 있는 것이며, 마찬가지로 subPropertyOf 추론은 어떤 속성 p_i가 p_j의 하위 속성일 경우($p_i \subset p_j$) p_i가 p_j로 해석될 수 있는 것이다. 또한 만약 $c_i \subset c_j$이면 c_i 및 c_i의 인스턴스는 c_j의 모든 속성을 계승한다.

• subClassOf의 추론에 의한 권한 전파

1) $c_i \subset c_j$, ca_j+ -> ca_i+, ca_i+' => ca_j+':

그림 3(a)에서 c_i가 c_j의 하위 클래스이므로 *type*이 R인 어떤 접근 권한 ca_j의 명세에 의한 ca_j+ -> ca_i+의

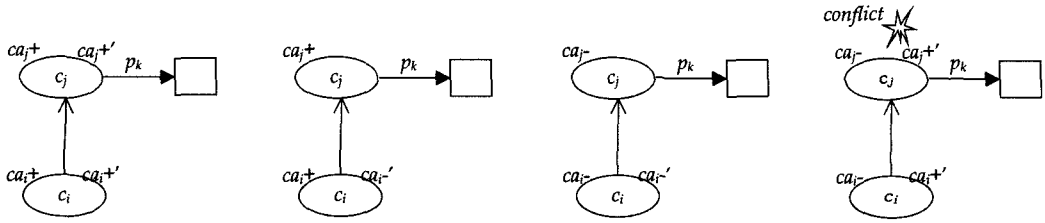


그림 3 subClassOf 추론에서의 권한 전파 충돌의 유형

권한 전파를 고려할 수 있다. 예로, ca_j+ = <Dave, [Inpatient, SSN, $$z$], read, +, R>이 명세될 경우 하위 클래스로의 권한 전파에 의하여 ca_j+는 ca_i+ = <Dave, [IsolatedPatient, SSN, $$z$], read, +, R>를 함께 내포한다.

다음으로 이후 RDF 트리플 t_i = [c_i, p_k, $$z$]에 새롭게 명세된 ca_i+'의 접근 권한을 고려하자. subClassOf 추론에 의하여 t_i로부터 t_j = [c_j, p_k, $$z$]를 해석할 수 있으므로, 추론에 의한 권한 전파 ca_i+' => ca_j+'를 고려할 수 있다. 이 경우 $sign(ca_j+')$ ≡ $sign(ca_j+)$ 이므로 충돌이 발생하지 않는다. 예로, ca_i+' = <Dave, [IsolatedPatient, SSN, $$z$], +, R>이 새롭게 명세될 경우 [IsolatedPatient, SSN, $$z$]로부터 [Inpatient, SSN, $$z$]가 추론 될 수 있으므로, 추론에 의한 권한 전파 ca_j+' = <Dave, [Inpatient, SSN, $$z$], read, +, L>을 고려할 수 있다. $sign(ca_j+')$ ≡ $sign(ca_j+)$ 이므로 충돌이 발생하지 않는다.

2) $c_i \subset c_j$, ca_j+ -> ca_i+, ca_i-' ≠> ca_j-':

마찬가지로, 그림 3(b)의 권한 전파에 의한 ca_j+ -> ca_i+ 권한 전파를 고려하자. 또한 이후 [c_i, p_k, $$z$]에 별도로 명세된 ca_i-'의 접근 권한을 고려하자. 이 경우, $sign(ca_i-')$ = -이므로 클래스 c_i의 속성 p_k에 대한 접근이 허용되지 않게 되므로 c_j로의 subClassOf 추론은 당연히 수행될 수 없다. 따라서 추론에 의한 권한 전파 충돌은 발생하지 않는다.

3) $c_i \subset c_j$, ca_j- -> ca_i-, ca_i-' ≠> ca_j-':

그림 3(c)의 ca_j- -> ca_i- 권한 전파를 고려하자. 또한 이후 [c_i, p_k, $$z$]에 별도로 명세된 ca_i-'의 접근권한을 고려하자. 이 경우 역시, 클래스 c_i의 속성 p_k에 대한 접근이 허용되지 않게 되므로 c_j로의 subClassOf 추론은 수행될 수 없다. 따라서 추론에 의한 권한 전파 충돌은 발생하지 않는다.

4) $c_i \subset c_j$, ca_j- -> ca_i-, ca_i+' => ca_j+':

그림 3(d)의 ca_j- -> ca_i- 권한 전파를 고려하자. 또한, 이후 [c_i, p_k, $$z$]에 새롭게 명세된 ca_i+'의 접근 권한을 고려하자. 이 경우 subClassOf 추론에 의한 권한

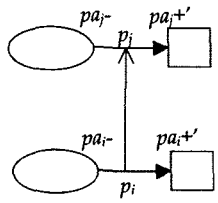


그림 4 subPropertyOf 추론에서의 권한 전파 충돌

전파 ca_i+' => ca_j+'를 고려할 수 있으므로, $sign(ca_j+')$ ≠ $sign(ca_j-)$의 충돌이 발생한다. 따라서 이 경우, ca_i+'는 허용되어서는 안 된다. 마찬가지로, ca_j+ -> ca_i+가 명세되어 있고 이후 *type* 값이 L인 ca_j- 접근 권한을 다시 명세할 경우, 충돌을 발생시키므로 이것 또한 허용되어서는 안 된다.

• subPropertyOf의 추론에 의한 권한 전파

*subClassOf*와 마찬가지로, 그림 4와 같이 $p_i \subset p_j$, pa_j- -> pa_i-, pa_i+' => pa_j+'의 경우 충돌이 발생하며, 따라서 이러한 권한 설정은 허용되어서는 안 된다.

5. 소수 그래프 레이블링을 통한 효율적 권한 충돌 발견

Jain과 Farkas[4]는 접근 권한들 사이에서의 추론에 의한 충돌을 발견하기 위하여, 어떤 접근 권한이 주어질 경우 주어진 접근 권한에 따라 먼저 모든 인스턴스에 $sign \in \{+, -\}$ 값을 할당한다. 다음으로 할당된 *sign* 값과 이전 *sign* 값 사이의 추론에 의한 충돌이 존재하는지를 모든 인스턴스에 대하여 조사하는 방식이다. 따라서 인스턴스 수가 많아지면 비효율적이다. 본 연구에서는 그래프 레이블링 기법[9,13]을 이용하여 조상/후손(ancestor/descendant) 관계에 있는 클래스 혹은 속성에 관한 접근 권한만을 검증함으로서, 효율적으로 접근 권한의 충돌을 발견하는 방법을 제안한다. 그래프 레이블링 기법으로는 프리픽스(prefix) 기법[13], 인터벌(interval) 기법[13], 소수(prime number)기법[9]등이 있지만, 가장 효율적인 방법으로 생각되는 소수 기법을 이용한다. 그 이유는 프리픽스, 인터벌 기법의 경우 조상/후손

관계를 알기 위한 연산이 한 번에 이루어지는 것이 아니라 반복적으로 이루어지기 때문에 그 비용이 크다[9]. 기본적인 아이디어는 4.2절에서의 "$c_i \subset c_j$, ca_j^- -> ca_i^-, $ca_i^{+\prime}$ => $ca_j^{+\prime}$" 와 "$p_i \subset p_j$, pa_j^- -> pa_i^-, $pa_i^{+\prime}$ => $pa_j^{+\prime}$"의 관찰에 의하여, +의 접근권한이 설정될 경우 조상 관계에 있는 - 접근 권한의 존재 여부를 조사하는 것이고, 반대로 *type* L의 -의 접근 권한이 설정될 경우 후손 관계에 있는 + 접근 권한의 존재 여부를 조사하는 것이다. 이러한 조상/후손 관계의 파악을 위하여 그래프 레이블링 기법을 이용한다.

5.1 소수 기반 그래프 레이블링

제안하는 방법에서는 소수 기반 그래프 레이블링을 subClassOf 관계와 subPropertyOf 관계로 나누어 수행한다. 먼저 subClassOf 관계에 대한 그래프 레이블링을 소개한다. 그림 5의 클래스 계층 구조에서 *a* 노드와 같이 모든 클래스들이 계승하는 루트(root) 클래스가 하나 존재하는, 하나의 연결 컴포넌트(connected component)로 구성된 방향성 비순환 그래프(Directed Acyclic Graph, DAG)를 고려할 수 있다. 만약 그래프가 하나의 연결 컴포넌트(connected component)가 아니면, 즉 포레스트(forest)이면, 가상의 루트 노드를 만들면 된다.

정의 3 (소수 기반 그래프 레이블링).

$G=(V, E)$를 DAG라고 하자. 만약 너비 우선 탐색(breadth first search)에 따른 정점 $v \in V$에 일련의 소수 할당 $pnum(v)$를 고려할 때, 다음과 같은 레이블링 $label(v)$를 수행한다.

$$label(v) = pnum(v) \cdot \begin{cases} \prod_{w \in parents(v)} label(w), & in\text{-}degree(v) > 0 \\ 1 & in\text{-}degree(v) = 0 \end{cases} \quad (1)$$

예로, *a* 노드의 경우 진입 차수(in-degree)가 0이므로 소수 1이 할당되고, *g* 노드의 경우 *b, f, i* 노드로부터의 레이블 값이 자신의 소수 7과 곱해진다.

정리 1. G의 두 정점 v, w에 대하여 $w = ancestor(v)$를 만족할 때, $label(v)$ *mod* $label(w) = 0$이다.

증명. (1) 1을 포함한 임의의 소수들(중복 허용)의 곱

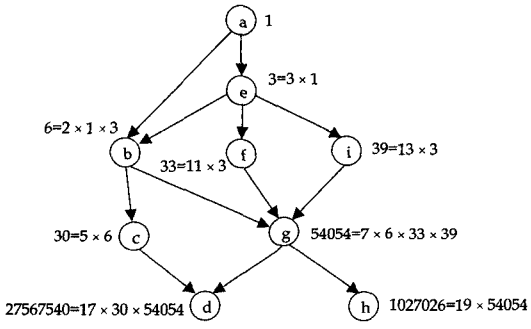


그림 5 소수 기반 DAG 레이블링

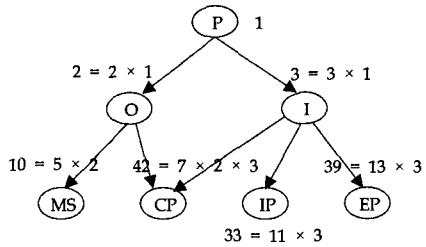


그림 6 샘플 RDFS에 대한 subClassOf 소수 기반 그래프 레이블링

$label(v) = 1 \cdot pnum_1 \cdot pnum_2 \cdot pnum_3 \cdot \ldots \cdot pnum_i$를 고려하자, $1 \le i \le k$. 또한 어떤 소수 $pnum_j$, $k < j$를 고려하자. $pnum_j$는 $label(v)$의 약수가 아니므로, $label(v)$ mod $pnum_j \neq 0$이다. 따라서 $label(w)$은 $label(v)$에 포함된 소수들의 곱으로 이루어져야한다. (2) 정의 1에 의하여 $label(v)$는 $label(w)$의 소수들의 곱을 포함한다. (1)과 (2)에 의하여 $label(v)$ *mod* $label(w) = 0$이다.

정리 1에 의하여 소수 레이블 값을 사용하여 클래스 계층 구조에서 임의의 두 클래스의 조상/후손 관계를 쉽게 파악할 수 있다. 그림 6은 그림 1의 RDFS 그래프에서 subClassOf의 관계에 의한 방향성 비순환 그래프와 소수 그래프 레이블 값을 보여준다(편의상 클래스 이름을 대문자로 약하여 표시하였음).

마찬가지로, subPropertyOf 관계에 의한 속성들의 DAG 그래프를 고려할 수 있으며, 그림 7은 그림 1에서의 subPropertyOf와 관련된 속성들의 그래프 레이블링을 보여준다.

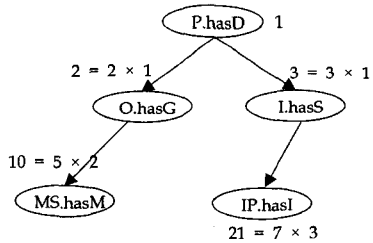


그림 7 샘플 RDFS에 대한 subPropertyOf 소수 기반 그래프 레이블링

5.2 레이블 값을 갖는 접근 권한의 저장

접근 권한은 그림 8의 관계형 테이블들에 저장된다. ATable은 원래의 명세된 접근 권한을 저장하며, CTable과 PTable은 권한 충돌의 효율적 발견을 위해 접근 권한의 파싱(parsing) 정보를 저장한다. CTable과 PTable의 각 필드에 빠른 탐색을 위한 인덱스가 설정될 수 있다. CTable은 subClassOf와 연관되는 접근 권한의 파싱 정보를 저장하며, PTable은 subPropertyOf와 연관

되는 접근 권한의 파싱 정보를 저장한다. *label* 필드는 소수 레이블 값을 저장하며, *r* 필드는 정의 1의 보안 객체 [*r*, *p*, *v*]에서의 *r* 값의 유형이 클래스 URI 상수(= *C*)인지 혹은 인스턴스 URI 상수(= *I*)인지를 저장하며, *p* 필드는 *p* 값을 저장한다. *sign* 필드는 접근 권한에서의 *sign* 값을 저장하며, *authorizationID* 필드는 ATable에서의 해당 접근 권한의 ID 값을 참조한다.

정의 4 (접근 권한 그래프 레이블 할당).

어떤 접근 권한 a_i는 기본적으로 subClassOf 레이블링 값 $label_{subClassOf}()$를 가지며, 보안 객체 [*r*, *p*, *v*]에서 속성 *p*가 subPropertyOf와도 연관될 경우 subPropertyOf 레이블링 값 $label_{subPropertyOf}()$를 별도로 갖는다. 먼저 subClassOf와 관련하여, 1) *r*이 클래스 URI 상수이고 *p*의 조건을 만족할 경우 해당 클래스의 subClassOf 레이블링 값 $label_{subClassOf}(r)$를 갖는다. 2) *r*이 인스턴스일 경우는 해당 클래스의 레이블링 값을 갖는다. 3) 만약 *r*이 $x와 같은 변수이면 *p*를 만족하는 subClassOf DAG에서의 최상위 클래스의 레이블 값을 갖는다. 다음으로 subPropertyOf 레이블 할당과 관련하여, [*r*, *p*, *v*]와 매치되는 subProperty DAG에서의 $label_{subPropertyOf}(p)$를 갖는다. 예외적으로, [$x, $y, $z]의 경우 모든 접근 권한과의 충돌을 조사해야 하므로, $label_{subClassOf}() = 1$ 그리고 $label_{subPropertyOf}() = 1$ 로 정의한다.

예로, 그림 8(c)의 접근 권한 R2에 대하여, *r* = IsolatedPatient는 클래스 URI 상수이고 *p* = $y로서 특정 속성을 갖는 클래스로 제한하지 않으므로 $label_{subClassOf}$(IsolatedPatient)는 그림 6에서 33이다. 이것은 CTable에 <33, C, $y, +, R2>의 튜플로 저장된다. R4의 경우 Shrek은 Outpatient의 인스턴스이므로 Outpatient의 레이블 값 2를 갖는다. *r*이 $x인 R6의 경우 속성 계승(inheritance)에 의하여 hasSpecialDisease 속성을 갖는 클래스는 Inpatient, EmergencyPatient, IsolatedPatient 이지만, 최상위 클래스는 Inpatient이므로 R6는 Inpatient의 레이블 값 3을 갖는다. 다음으로 PTable에서 R2의 경우 *r* = IsolatedPatient, *p* = $y는 그림 7 subPropertyOf DAG에서 IsolatedPatient.hasIsolationDisease를 포함하므로, 레이블 값 21이 할당된다. R5의 경우는 RDF 패턴 [EmergencyPatient, $y, $z]과 매치되는 것이 그림 7의 subPropertyOf DAG에 존재하지 않으므로 PTable에 저장되지 않는다.

5.3 권한 충돌 발견 알고리즘

제안하는 알고리즘은 그림 9(a)(b)의 권한 충돌 판정표에 의하여 충돌 가능의 경우만 충돌이 발생하는 지를 조사하며, 나머지의 경우는 무조건 충돌로 판정함으로서 효율적 권한 충돌 발견을 수행한다.

그림 9의 권한 충돌 판정표는 정의 1의 RDF 패턴, 즉 접근 권한 보안 객체에서의 *p*의 유형 ∈ {$y, URI 상수}에 따른 조상/후손 관계의 클래스 혹은 속성에서의 충돌 판정을 정리한다. 이를 앞서 그림 8에서의 접근 권한 예를 통하여 설명한다.

경우 그림 9(a) - 1 : 접근 권한 R7 = <Dave, [MedicalSubject, $y, $z], read, +, L>이 새롭게 명세될 경우, MedicalSubject의 조상 클래스와 관련한 충돌 접근 권한은 R1이다. R4와 충돌이 발생지 않는 이유는 잠시 후 설명한다. *p*(R7) = $y, *p*(R1) = $y, *sign*(R7) = +, *sign*(R1) = - 이므로, 이것은 그림 9(a) 표의 1)에 의하여 무조건 충돌이다. 왜냐하면 *p*(R7) = $y는 Outpatient로부터 계승되는 모든 속성을 포함하는 것을 의미하기 때문이다. 다시 접근 권한 R7 = <Dave, [Pinocchio, $y, $z], read, +, L>가 새롭게 명세되는 경우를 가정하자.

label	*r*	*p*	*sign*	*authorizationID*
2	C	$y	-	R1
33	C	$y	+	R2
3	C	hasSpecialDisease	+	R3
2	I	$y	-	R4
39	C	$y	+	R5
3	C	hasSpecialDisease	+	R6
...	...	...	...	...

(a) CTable

label	*r*	*p*	*sign*	*authorizationID*
2	C	$y	-	R1
21	C	$y	+	R2
3	C	hasSpecialDisease	+	R3
2	I	hasGeneralDisease	-	R4
3	C	hasSpecialDisease	+	R6
...	...	...	...	...

(b) PTable

ID	Authorization
R1	<Dave, [Outpatient, $y, $z], read, -, R>
R2	<Dave, [IsolatedPatient, $y, $z], read, +, L>
R3	<Dave, [Inpatient, hasSpecialDisease, $z], read, +, R>
R4	<Dave, [Shrek, $y, $z], read, -, R>
R5	<Dave, [EmergencyPatient, $y, $z], read, +, R>
R6	<Dave, [$x, hasSpecialDisease, $z], read, +, R>
...	...

(c) ATable

그림 8 접근 권한 저장 테이블

조상(-) / 후손(+)	$y	URI 상수
$y	1)충돌	2)충돌
URI 상수	3)충돌 가능	4)충돌 가능

(a) subClassOf 관계

조상(-) / 후손(+)	$y	URI 상수
$y	1)충돌	2)충돌
URI 상수	3)충돌	4)충돌

(b) subPropertyOf 관계

그림 9 속성 p의 유형 ∈ {$y, URI 상수}에 따른 권한 충돌 판정

즉, RDF 패턴에서의 r 값이 인스턴스 URI 상수인 경우를 가정하자. 모든 인스턴스에서 Pinocchio의 URI 상수 값을 갖는 인스턴스는 유일하므로(unique), 조상과의 관계를 갖는(후손 관계는 해당 안됨) 접근 권한에서 r의 유형이 인스턴스가 아닌 클래스인 경우만이 R7에 영향을 준다. 예로, R4의 경우 비록 p(R7) = $y, p(R4) = $y, $sign$(R7) = +, $sign$(R4) = - 이지만, r 값이 Shrek에 관한 것이므로 접근 권한 충돌이 발생하지 않는다. 후손 관계는 해당 되지 않음에 대하여, 접근 권한 R7 = <Dave, [MedicalSubject, $y, $z], read, +, L>을 다시 가정하자. 비록 R4는 그림 9(a) 표의 1)에 의하여 R7과 후손 관계의 충돌을 가질 수 있지만, Shrek은 Outpatient의 유일한 인스턴스이고 또한 Outpatient로부터 MedicalSubject로의 추론은 일어날 수 없으므로, 어떤 권한의 r 값이 인스턴스 URI이면 후손 관계로 부터의 추론에 의한 충돌은 발생할 수 없다.

경우 그림 9(a) - 2 : 접근 권한 R7 = <Dave, [Inpatient, SSN, $z], read, -, L>이 명시된다고 하자. Inpatient의 후손 클래스와 관련한 접근 권한은 R2와 R5인데, R2의 경우 p(R7) = SSN, p(R2) = $y, $sign$(R7) = -, $sign$(R2) = + 이므로 그림 9(a) 표의 2)에 의하여 무조건 충돌이다. 왜냐하면 p(R2) = $y는 Inpatient로부터 계승되는 모든 속성, 즉 SSN 속성을 포함하기 때문이다. 마찬가지로 R5의 경우도 R7과 충돌이다.

경우 그림 9(a) - 3: 다시 접근 권한 R7 = <Dave, [MedicalSubject, SSN, $z], read, +, L>를 고려하자. p(R7) = SSN, p(R1) = $x, $sign$(R7) = +, $sign$(R1) = - 이므로, 이것은 그림 9(a) 표의 3)에 의하여 충돌 가능의 경우이다. p(R7) = SSN은 Outpatient로부터 계승되기 때문에 최종 판정은 충돌이다. 하지만, 그림 9(a) 표의 3)의 경우 p 값 즉 URI 상수가 조상 클래스로부터 계승되지 않는 속성이면 충돌이 발생하지 않게 되므로, URI 상수가 계승되는지 여부를 조사할 필요가 있다.

경우 그림 9(a) - 4: 마찬가지로, 그림 9(a) 표의 4)의 경우도 두 URI 상수 값이 같은지 여부, 즉 두 접근 권한이 동일한 속성에 대하여 명세되는지를 조사할 필요가 있다. 만약 동일한 속성이면 충돌이고, 그렇지 않으면 충돌이 아니다.

경우 그림 9(b) 1 - 4 : 그림 9(b) 판정표의 subPropertyOf 관계에서는 모든 경우에 대하여 충돌이다. 왜냐하면 subClassOf 관계에서는 클래스의 계승 관계를 정의하기 때문에 경우 그림 9(a) - 3, 4와 같이 계승 관계에 있지 않은 속성이 존재할 수 있지만, subPropertyOf는 직접적으로 속성 사이의 계승 관계를 정의하기 때문이다. 예로 접근 권한 R7 = <Dave, [Patient, $y, $z], read, -, L> 가 명세됨을 가정하자. 그러면 R3와의 관계에서 p(R7) = $y, p(R3) = hasSpecialDisease, $sign$(R7) = -, $sign$(R3) = + 이므로 그림 9(a) - 3의 경우대로라면 충돌이 아닐 수도 있다. 하지만, 그림 8(b)의 PTable은 그림 7의 subPropertyOf 관계에 해당하는 속성과 관련한 접근 권한의 파싱 정보를 저장한 것이므로 항상 충돌이다. 그림 9(b)의 판정표는 이를 반영한다. 또한 R7 = <Dave, [Patient, hasDisease, $z], read, -, R> 가 명시될 경우, R7의 속성 URI 상수 hasDisease와 R3의 속성 URI 상수 hasSpecialDisease는 항상 그림 7의 subPropertyOf DAG에서의 조상/후손 관계를 만족하게 되므로 무조건 충돌이다. 이는 그림 9(b) - 4의 경우를 반영한다. 경우 그림 9(a) - 1에서의 관찰에 따라, 조상(-)에 해당하는 접근 권한의 r값의 유형이 인스턴스 URI 상수인 경우 충돌이 아니며, 나머지는 무조건 충돌이다.

그림 10의 알고리즘은 지금까지 설명한 조상(-)/후손(+) 관계 및, p의 유형 ∈ {$x, URI 상수}에 따른 권한 충돌 판정표를 반영하여 효율적으로 권한 충돌 발견을 수행한다. 권한 충돌 발견은 subClassOf와 관련한 CTable과 subPropertyOf와 관련한 PTable 로 각각 나누어 수행된다. 주어진 접근 권한 a_i가 $sign$ = +일 경우 라인(line) 3, 4, 27, 28에 의해 조상 관계의 접근 권한 중 $sign$ = -인 경우를 조사하고, a_i의 sign = -일 경우 라인 14, 15, 35, 36에 의해 후손 관계의 접근 권한 중 $sign$ = +인 경우를 조사한다. 그림 9(a)의 판정표의 3, 4의 경우에 따라, 라인 7과 18은 이를 조사하며, 나머지 경우는 모두 충돌이다. 즉, 속성이 계승되지 않거나 ($p(a_i) \notin properties(r(rs_i))$ 또는 $p(rs_i) \notin properties(r(a_i))$), 두 속성이 다른 경우($p(a_i) \neq p(rs_i)$) 충돌이 아니다. 또한 경우 그림 9(a) - 1에서의 관찰에 따라,

```
Algorithm find_AuthoConflict1
Input: 명세될 접근 권한 a_i
Output: 충돌 관계에 있는 접근 권한들의 ID 집합 Conflict_Set
1   L1 := label_subClassOf(a_i);
2   if (L1 = 1 ∧ (type(a_i) = R ∨ sign(a_i) = '+')) then Not Conflict; return Conflict_Set;
                          /* [$x, $y, $z]의 경우 혹은 r이 루트(root) 클래스일 경우는 충돌 발생 안함 */

    /* subClassOf 관계에 의한 조상(-)과의 충돌 여부 조사 */
3   if sign(a_i) = '+' then
4     RS := executeQuery ("select * from CTable where (L1 % label) = 0 and sign = '-'");
5   if RS is not empty then
6     foreach rs_i ∈ RS do   /* 판정표 9(a)의 3, 4 경우 조사 */
7       if ((p(a_i) ∈ {속성 URI 상수} ∧ p(rs_i) = $y ∧ p(a_i) ∉ properties(r(rs_i))) ∨
                                                  (p(a_i), p(rs_i) ∈ {속성 URI 상수} ∧ p(a_i) ≠ p(rs_i)))
8         Not Conflict; Continue;
9       else
10        if r(rs_i) ∈ {인스턴스 URI 상수} then Not Conflict; Continue; /* r(rs_i)가 인스턴스 URI인 경우 제외 */
11        else
12          Conflict;
13          Conflict_Set := Conflict_Set ∪ rs_i.authorization_ID

    /* subClassOf 관계에 의한 후손(+)과의 충돌 여부 조사 */
14  if sign(a_i) = '-' and type(a_i) = 'L' then
15    RS = executeQuery ("select * from CTable where (label % L1) = 0 and sign = '+'");
16  if RS is not empty then
17    foreach rs_i ∈ RS do  /* 판정표 9(a)의 3, 4 경우 조사 */
18      if ((p(rs_i) ∈ {속성 URI 상수} ∧ p(a_i) = $y ∧ p(rs_i) ∉ properties(r(a_i))) ∨
                                                  (p(a_i), p(rs_i) ∈ {URI 상수} ∧ p(a_i) ≠ p(rs_i)))
19        Not Conflict; Continue;
20      else
21        if r(a_i) ∈ {인스턴스 URI 상수} then Not Conflict; Continue;  /* r(a_i)가 인스턴스 URI인 경우 제외 */
22        else
23          Conflict;
24          Conflict_Set := Conflict_Set ∪ rs_i.authorization_ID

25  L2 := label_subPropertyOf(a_i);
26  if L2 = 1 then Not Conflict; return Conflict_Set;
          /* [$x, $y, $z]의 경우 혹은 p가 subPropertyOf DAG에서 최상위 속성일 경우는 충돌 발생 안함 */
    /* subPropertyOf 관계에 의한 조상(-)과의 충돌 여부 조사 */
27  if sign(a_i) = '+' then
28    RS := executeQuery ("select * from PTable where (L2 % label) = 0 and sign = '-'");
29  if RS is not empty then
30    foreach rs_i ∈ RS do  /* r(rs_i)가 인스턴스 URI 상수인 경우를 제외하곤 무조건 충돌 */
31      if r(rs_i) ∈ {인스턴스 URI 상수} then Not Conflict; Continue;
32      else
33        Conflict;
34        Conflict_Set := Conflict_Set ∪ rs_i.authorization_ID

    /* subPropertyOf 관계에 의한 후손(+)과의 충돌 여부 조사 */
35  if sign(a_i) = '-' and type(a_i) = 'L' then
36    RS := executeQuery ("select * from PTable where (label % L2) = 0 and sign = '+'");
37  if RS is not empty then
38    foreach rs_i ∈ RS do   /* r(a_i)가 인스턴스 URI 상수인 경우를 제외하곤 무조건 충돌 */
39      if r(a_i) ∈ {인스턴스 URI 상수} then Not Conflict; Continue;
40      else
41        Conflict;
42        Conflict_Set := Conflict_Set ∪ rs_i.authorization_ID

43  return Conflict_Set;
```

그림 10 제안하는 권한 충돌 발견 알고리즘

조상(-)에 해당하는 접근 권한의 r 값의 유형이 인스턴스 URI 상수인 경우 충돌이 아니다. 라인 10, 21, 31, 39는 이를 반영한다.

6. 성능 평가

본 절에서는 Jain과 Farkas의 모든 인스턴스에 대한 권한 충돌 조사 방법[4]에 대하여 본 연구에서 제안하는 소수 그래프 레이블링을 이용한 권한 충돌 방법이 어느 정도 효율적인지를 비교 분석한다.

6.1 실험 환경

Jain과 Farkas 방법의 최적화된 구현 소스를 구할 수 없었기 때문에 다음과 같은 모의실험을 수행하였다. 먼저 표 1에서의 실험 인자 #*C*와 #*P*, 그리고 #*S*에 따른 임의의 DAG를 랜덤하게 생성한다. 예로 그림 11에서 원은 클래스를 나타내며, 사각형은 각 클래스의 속성을 나타낸다. 또한 클래스 *c3*는 *c1*과 *c2*의 하위 클래스이므로, #S의 값은 2이다. 속성에 대한 자료 구조에 대하여 이전 권한 명세에 의한 *sign* 값을 저장하기 위한 *before_sign* 변수와 새로운 권한 명세에 의한 *sign* 값을 저장하기 위한 *after_sign* 변수가 각 속성에 대하여 생성된다. 또한 상위 클래스에 대한 계승 속성을 하위 클래스가 별도로 갖는다. 예로, *c1* 클래스의 속성 *p1*과 *c2* 클래스의 속성 *p2*를 *c3* 클래스는 별도로 갖는다.

Jain과 Farkas에 대한 모의 방법은 그림 12의 알고리즘과 같다. 먼저 현재까지 명세된 접근 권한 집합 *RS*의 모든 접근 권한 rs_i들에 대하여(이러한 접근 권한들은 충돌 관계에 있지 않음), 라인 1과 2에 의해 그림 11의 그래프를 너비 우선 탐색(breadth first search)하며 *before_sign* 변수에 권한 명세 전파에 의한 *sign* 값을

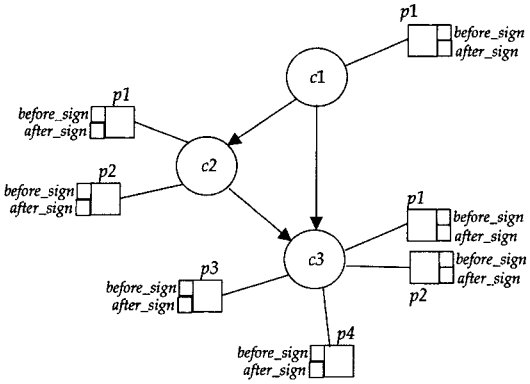


그림 11 모의실험을 위한 DAG 그래프 생성의 예

할당한다. 이 과정을 거치면 모든 *before_sign* 변수는 *RS*에 해당하는 *sign* 값을 할당받게 된다. 다음으로 새롭게 명세될 접근 권한 a_i에 대하여, 라인 3에 의해 그래프를 다시 너비 우선 탐색하며 *after_sign* 변수에 권한 명세 전파에 의한 *sign* 값을 할당한다. 마지막으로 *isConflict*() 함수에 의해, 그래프를 다시 깊이 우선 탐색하며, 속성의 *before_sign* 값이(-), 그리고 *after_sign* 값이(+)인 경우가 존재하는 지를 조사한다. 만약 존재하면 충돌로 판정하고 그렇지 않으면 비충돌로 판정한다. 본 실험에서는 Jain과 Farkas의 방법에 더 이점을 주기 위하여 너비 우선 탐색을 반복적인(iterative) 방법으로 구현하였다. 모든 실험은 RAM 1GB와 3.2 GHz 듀얼 펜티엄 IV CPU를 가진 윈도우 XP 컴퓨터에서 수행되었으며, 실험 프로그램은 자바 언어로 개발되었다.

6.2 실험 인자 #A에 따른 분석

실험 인자 #A에 따른 실험은 다음과 같이 수행되었

표 1 실험 인자

실험 인자	값의 범위	설명
#C	100~1,000	그래프에서의 노드의 수; 이것은 RDF 클래스의 수 혹은 인스턴스의 수로 해석될 수 있다.
#P	1~5	각 클래스의 평균 속성의 수
#S	1~5	각 클래스가 subClassOf 관계를 갖는 평균 수
#A	10~100, 200, 300, 400, 500	명세되는 접근 권한의 수

Algorithm **find_AuthoConflict2**
Input: 명세될 접근 권한 a_i, 기존의 명세된 접근 권한 집합 *RS*
Output: 충돌의 경우 true 반환, 충돌이 아닌 경우 false 반환

```
1    foreach rs_i ∈ RS do
2        autho_propagation_before(rs_i);   /* RS의 모든 rs_i에 대하여 before_sign 변수에 rs_i의 sign 값 할당 */
3    autho_propagation_after(a_i);   /* after_sign 변수에 a_i의 sign 값 할당 */
4    if isConflict() then return true;
5    else return false;
```

그림 12 Jain과 Farkas에 대한 모의 알고리즘

다. 먼저 생성된 *DAG*에 대하여 랜덤하게 임의의 접근 권한을 생성한다. 다음으로 생성된 접근 권한에 대하여, 그림 10과 그림 12의 알고리즘을 각각 수행하고, 만약 충돌이 없는 경우 각각의 *RS*에 저장한다. 이러한 과정을 *#A* 만큼 반복하였으며, 반복 횟수 동안의 수행 시간을 측정하였다. 이 실험에서의 또 하나의 중요한 관찰은 최종 수행 후 각각의 *RS*에 저장된 충돌 관계에 있지 않은 접근 권한들이 동일해야 한다는 것인데, 모든 반복 실험에서 이러한 정확성을 검증하였다.

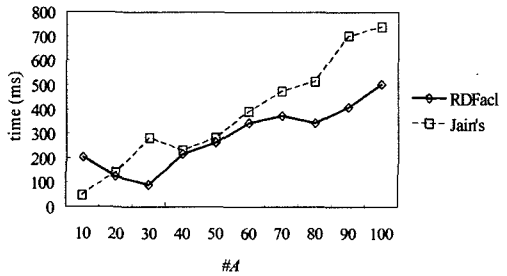
그림 13은 *#C*가 100, *#S*가 2일 때와 *#C*가 1,000, *#S*가 5일 때의 각각의 *#A*의 변화에 따른 권한 충돌 발견 시간을 보여 준다. 네 그래프 모두에서 제안된 방법이 매우 효율적인 권한 충돌 발견 시간을 가짐을 보여 주며, 또한 접근 권한의 수가 증가해도 Jain과 Farkas의 방법에 비교하여 그 증가율이 매우 작음을 보여 준다.

6.3 실험 인자 *#C*에 따른 분석

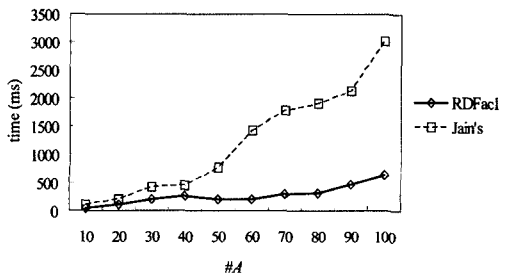
다음으로 그래프의 노드의 수, 즉 RDF 클래스의 수 혹은 인스턴스의 수의 증가에 따른 권한 충돌 발견 시간을 비교하였다. 6.2절의 실험에서와 마찬가지로 랜덤하게 발생되는 새로운 접근 권한들의 전체 충돌 발견 시간을 측정하였다. 그림 14는 *#A*가 100, *#S*가 2일 때와 *#A*가 500, *#S*가 5일 때의 *#C*의 변화에 따른 권한 충돌 발견 시간을 보여 준다. 두 그래프 모두 제안된 방법이 매우 효율적인 권한 충돌 발견 시간을 가짐을 보여 준다. 이러한 결과는 사실 모든 RDF 인스턴스를 조사하는 Jain과 Farkas의 방법에 비교하여, 본 연구에서 제안된 방법이 소수 그래프 레이블링을 이용하여 충돌 관계에 있는 몇몇 접근 권한만을 조사하므로 당연하다.

6.4 저장 비용 분석

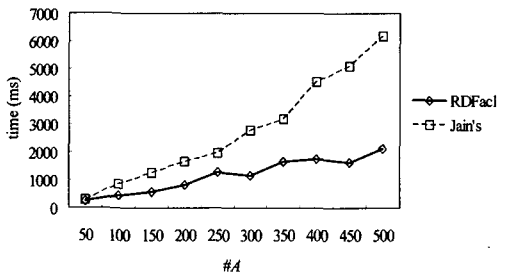
제안된 방법은 권한 충돌의 효율적 발견을 위하여 그



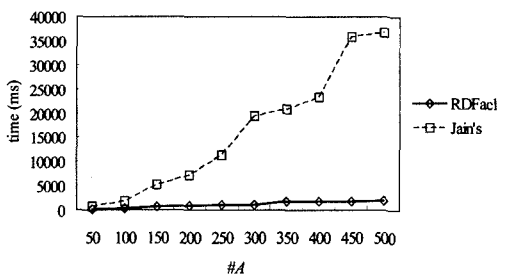
(a) *#C* = 100, *#S* = 2, *#A*: 10~100



(b) *#C* = 1000, *#S* = 5, *#A*: 10~100

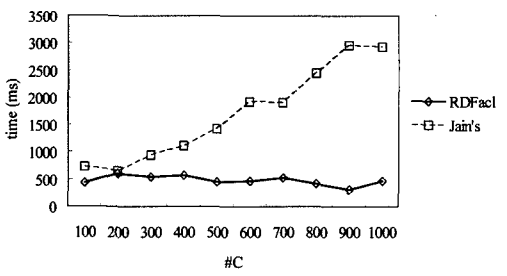


(c) *#C* = 100, *#S* = 2, *#A*: 100~500

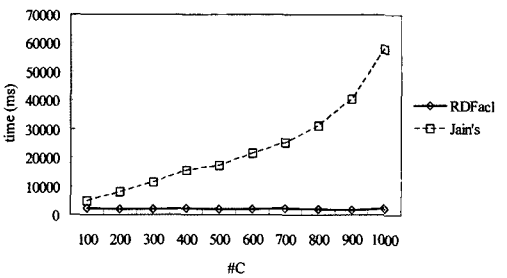


(d) *#C* = 1000, *#S* = 5, *#A*: 100~500

그림 13 *#A*에 따른 접근 권한 충돌 발견 시간 비교



(a) *#A* = 100, *#S* = 2, *#C*: 100~1000



(b) *#A* = 500, *#S* = 5, *#C*: 100~1000

그림 14 *#C*에 따른 접근 권한 충돌 발견 시간 비교

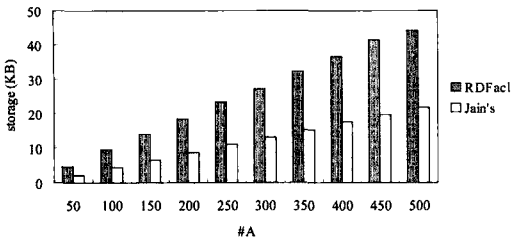


그림 15 #*A*에 따른 권한 저장소의 저장 크기 비교

림 8에서 ATable 이외에 CTable과 PTable을 사용하므로 기존의 방법보다 별도의 저장 공간을 더 요구하게 된다. 하지만 이러한 추가 저장 공간은 앞서 보여준 수행 성능 시간의 향상에 비교한다면 미미하다고 판단한다. 그림 15의 그래프는 #*A*에 따른 저장 공간의 사용량을 보여준다. RDFacl이 약 2배의 저장 공간을 사용하는 것을 보여 준다.

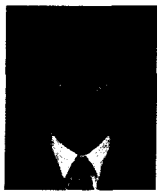
7. 결론 및 향후 연구

RDF 권한 충돌 문제는 RDF 접근 제어 명세에서 중요한 문제이다. 왜냐하면 RDF 데이타는 XML 데이타와 달리 RDF 추론 기능이 존재하기 때문이다. 따라서 본 논문에서는 먼저 RDF 트리플에 기반한 접근 권한 명세 모델 및 RDF 추론시의 권한 충돌 문제를 간단히 소개하였다. 다음으로 이러한 분석을 토대로, 권한 충돌을 효율적으로 발견하기 위한, 소수 그래프 레이블링 기법을 이용하는 새로운 권한 충돌 방법을 자세히 소개하였다. 제안된 방법은 그래프 레이블링 값을 이용하여 조상/후손 관계의 충돌의 가능성이 있는 권한들만을 선별적으로 조사한다. 이러한 방법은 RDF 추론에서의 핵심이 되는 포함 관계 기반 추론(subClassOf 및 subPropertyOf)에 있어 매우 효율적일 것이다.

앞으로의 연구 방향은 본 연구에서의 RDF 접근 제어 모델을 보다 복잡한 온톨로지 어휘 및 추론을 제공하는 OWL에 대하여 확장하는 것이다. 또한 RDF 문서의 암호화에 대하여, 제안하는 RDF 접근 제어 모델을 적용하여 세밀한(fine-grained) 암호화 과정을 효율적으로 수행하는 방법에 대하여 추가적으로 연구하고자 한다. 본 연구에서 고려한 정의 2에서의 읽기 *action*외에 변경, 생성, 삭제등과 같은 쓰기 *action*에서의 추론에 의한 권한 충돌 문제를 살펴보는 것도 또한 본 연구의 향후 과제이다.

참 고 문 헌

[1] Semantic Web, W3C, http://www.w3.org/2001/sw
[2] RDF Primer, W3C Recommendation, http://www.w3.org/TR/rdf-primer/
[3] OWL Web Ontology Language Overview, W3C Recommendation, http://www.w3.org/TR/owl-features/
[4] A. Jain, C. Farkas, "Secure resource description framework: an access control model," Proc. of 11th ACM Symposium on Access Control Models and Technologies, pp. 121-129, June 2006.
[5] L. Qin, V. Atluri, "Concept-level Access Control for the Semantic Web," Proc. of ACM Workshop on XML Security 2003, pp. 94-103, Oct. 2003.
[6] S. Javanmardi, M. Amini, R. Jalili, "An Access Control Model for Protecting Semantic Web Resources," Proc. of the 2nd International Semantic Web Policy Workshop(SWPW'06), Nov. 2006.
[7] S. Kaushik, D. Wijesekera, P. Ammann, "Policy-based dissemination of partial web-ontologies," Proc. of the 2005 workshop on Secure web services, pp. 43-52, Nov. 2005.
[8] P. Reddivari, T. Finin, A. Joshi, "Policy-Based Access Control for an RDF Store," Proc. of the Policy Management for the Web Workshop, pp. 78-83, May. 2005.
[9] G. Wu, K. Zhang, C. Liu, J. Li, "Adapting Prime Number Labeling Scheme for Directed Acyclic Graphs," DASFAA 2006, pp. 787-796, April 2006.
[10] E. Damiani, S. D. C. Vimercati, S. Paraboschi, P. Samarati, "A fine-grained access control system for XML documents," ACM Transactions on Information and System Security, 5(2), pp. 169-202, 2002.
[11] E. Bertino, S. Castano, E. Ferrari, M. Mesiti, "Specifying and enforcing access control policies for XML document sources," World Wide Web Journal, 3(3), pp. 139-151, 2000.
[12] E. Bertino, E. Ferrari, "Secure and selective dissemination of XML documents," ACM Transactions on Information and System Security, 5(3), pp. 290-331, 2002.
[13] V. Christophides, G. Karvounarakis, D. Plexousakis, M. Scholl, S. Tourtounis, "Optimizing taxonomic semantic web queries using labeling schemes," Journal of Web Semantics, 11(1), pp. 207-228, Nov. 2003.
[14] R. Agrawal, A. Borgida, and H. V. Jagadish, "Efficient management of transitive relationships in large data and knowledge bases," In Proc. of the SIGMOD Inter. Conf. on Management of Data, pp. 253-262, 1989.
[15] Q. Li and B. Moon, "Indexing and querying XML data for regular path expressions," In Proc. of 27th Inter. Conf. on Very Large Data Bases(VLDB'02), pp. 361-370, 2001.
[16] Online Computer Library Center. Dewey decimal classification. http://www.oclc.org/dewey
[17] RDF Semantics, W3C Recommendation, http://www.w3.org/TR/rdf-mt/



김 재 훈

1997년 건국대학교 전자계산학과 공학사 1999년 건국대학교 컴퓨터·정보통신공학과 공학석사. 2005년 서강대학교 컴퓨터학과 공학박사. 2005년 3월~2006년 9월 삼성전자 정보통신총괄 통신연구소 책임연구원. 2006년 9월~현재 서강대학교 컴퓨터공학과 BK 21 계약교수. 관심분야는 시맨틱 웹, 웹 데이타베이스, 데이타베이스 보안임

박 석

1978년 서울대학교 계산통계학과(이학사) 1980년 한국과학기술원 전산학과(공학석사). 1983년 한국과학기술원 전산학과(공학박사). 1983년 9월~현재 서강대학교 컴퓨터공학과 교수. 1989년~1991년/2002년~2003년 University of Virginia 방문교수. 1997년 2월~현재 한국정보보호학회 이사. 2005년 한국정보과학회 부회장. 2004년 1월~2005년 12월 한국정보과학회 편집위원장. 1999년~2007년 DASFAA Steering Committee 멤버. 관심분야는 데이타베이스 보안, 실시간 시스템, 트랜잭션 관리, 데이타웨어하우스, 웹 데이타베이스임