온톨로지 디버깅을 위한 종속 부호 기반 비논리적 공리 탐지

(Dependency Label based Causing Inconsistency Axiom Detection for Ontology Debugging)

김 제 민 † 박 영 택 ††
(Je-Min Kim) (Young-Tack Park)

요 약 W3C는 시맨틱 웹 환경에서 온톨로지를 저작하고 공유하기 위해 온톨로지 구축 언어인 OWL을 발표하였다. 현재 OWL 온톨로지의 논리적 정당성을 검사하기 위해서, OWL 추론 엔진들이 소개되고 있다. 그러나 대부분의 추론 엔진들은 정당하지 못한 개념의 탐지 과정 없이 결과만을 보여준다. 본 논문에서는 온톨로지내의 정당하지 못한 개념을 디버깅하기 위해 종속 부호 기반 비논리적 공리(CIA-Causing Inconsistency Axiom) 탐색 기법을 제안한다. 비논리적 공리는 정당하지 못한 개념들을 유발하는 공리들의 집합이다. 비논리적 공리를 탐지하기 위해서는 온톨로지 내에서 비일관성을 유발하는 공리를 찾아내야 한다. 온톨로지 저작 도구에 정확한 비논리적 공리가 제공된다면, 온톨로지 저작 도구는 온톨로지 내에서의 정당하지 못한 내용을 수정할 수 있도록 수정될 일부 내용만을 보여줄 것이다. 따라서 본 논문은 두 부분에 초점을 맞추었다. 첫 번째, 정당하지 못한 내용을 가진 온톨로지가 주어졌을 때 비정당성을 유발하는 공리들을 도출하고, 이들의 근원을 식별한다. 두 번째 비정당성을 유발하는 공리가 탐지되었을 때 이들만을 추출하여, 온톨로지 설계자에게 보여주는 것 이다. 따라서 먼저 기존에 발표되었던 테이블로 알고리즘 기반의 결정 모듈을 소개하고, 이보다 향상된 기법인 종속 부호 기반 비논리적 공리 탐지 기법을 제안한다. 본 논문의 결과물은 현재 온톨로지 언어의 기본이 되는 SHOIN 서술 논리 응용시스템에 적용가능하다.

키워드 : 종속 부호, 온톨로지 디버깅, 온톨로지 추론

Abstract The web ontology language(OWL) has become a W3C recommendation to publish and share ontologies on the semantic web. In order to check the satisfiablity of concepts in OWL ontology, OWL reasoners have been introduced. But most reasoners simply report check results without providing a justification for any arbitrary entailment of unsatisfiable concept in OWL ontologies. In this paper, we propose dependency label based causing inconsistency axiom (CIA) detection for debugging unsatisfiable concepts in ontology. CIA is a set of axioms to occur unsatisfiable concepts. In order to detect CIA, we need to find axiom to cause inconsistency in ontology. If precise CIA is gave to ontology building tools, these ontology tools display CIA to debug unsatisfiable concepts as suitable presentation format. Our work focuses on two key aspects. First, when a inconsistency ontology is given, it detect axioms to occur unsatisfiable and identify the root of them. Second, when particular unsatisfiable concepts in an ontology are detected, it extracts them and presents to ontology designers. Therefore we introduce a tableau-based decision procedure and propose an improved method which is dependency label based causing inconsistency axiom detection. Our results are applicable to the very expressive logic *SHOIN* **that** is the basis of the Web Ontology Language.

Key words : Dependency Label, Ontology Debugging, Ontology Reasoning

· 본 논문은 숭실대학교의 지원을 받았습니다.

† 학생회원 : 숭실대학교 컴퓨터학과
kimjemins@hotmail.com
†† 종신회원 : 숭실대학교 컴퓨터학과 교수
park@ssu.ac.kr

논문접수 : 2007년 7월 3일
심사완료 : 2008년 10월 28일

정보과학회논문지: 소프트웨어 및 응용 제35권 제12호(2008.12)

1. 서 론

차세대 웹 환경으로 떠오르고 있는 시맨틱 웹의 주된 목적은 웹에 존재하는 정보를 기계가 이해하고 처리할 수 있도록 하는 것이다. 온톨로지는 시맨틱 웹이 정형화된 개념 정의와 공유를 통해서 의미 있는 정보를 가질 수 있도록 중추적인 역할을 한다. 웹 온톨로지 구축 언어인 OWL에 기본이 되는 서술 논리(Description Logics)는 온톨로지 개발자와 사용자에게 폭넓게 인지되고 있다. 그러나 복잡하게 구성된 온톨로지의 경우 서술 논리 표현에 익숙지 않거나 전문가가 아니라면, 온톨로지 구축 과정에서 발생하는 오류를 디버깅 하는 것은 매우 어려운 작업이 된다. 온톨로지의 논리적 정당성을 검사하기 위해서, OWL 추론 엔진들이 소개되고 있다. 그러나 대부분의 추론 엔진들은 정당하지 못한 개념의 탐지 과정 없이 결과만을 보여준다.

본 논문에서 온톨로지내의 정당하지 못한 개념을 디버깅하기 위해 종속 부호 기반 비논리적 공리(CIA-Causing Inconsistency Axiom) 탐색 기법을 제안한다. 비논리적 공리는 정당하지 못한 개념들을 유발하는 공리들의 집합이다. 온톨로지 저작 도구에 정확한 비논리적 공리가 제공된다면, 온톨로지 저작 도구는 온톨로지 내에서의 정당하지 못한 내용을 수정할 수 있도록 수정될 일부 내용만을 보여줄 것이다. 이러한 비논리적 공리를 탐지하기 위해서는 온톨로지 내용의 비일관성을 유발하는 공리들을 찾아내야 한다. 따라서 본 논문에서는 비일관성 유발에 원인을 제공하는 공리를 결정하는 종속 부호를 정의하고, 이를 바탕으로 비일관성을 유발하는 공리의 집합들을 최소화함으로써 온톨로지 개발자가 보다 쉽게 디버깅하도록 유도한다. 종속 부호를 효율적으로 구축하기 위해서, 제안한 기법은 테이블로 알고리즘을 확장하고, 몇 가지 명령어를 정의했다.

본 논문은 두 부분에 초점을 맞추었다. 첫 번째, 정당하지 못한 내용을 가진 온톨로지가 주어졌을 때 비정당성을 유발하는 공리들을 도출하고, 이들의 근원을 식별한다. 두 번째 비정당성을 유발하는 공리가 탐지되었을 때 이들만을 추출하여, 온톨로지 설계자에게 보여주는 것 이다. 따라서 먼저 기존에 발표되었던 테이블로 알고리즘 기반의 결정 모듈을 소개하고, 이보다 향상된 기법인 종속 부호 기반 비논리적 공리 탐지 기법을 제안한다.

본 논문에서 제안한 온톨로지 디버깅을 위한 종속 부호 기반 비논리적 공리 탐색 기법은 OWL-DL로 작성된 온톨로지의 모든 결점들을 파악하는데 도움을 준다. 따라서 본 논문의 결과물은 서술 논리 기반의 지식 설명 서비스(DL Explanation Service), 디버깅 서비스(DL Debugging Service), 수정 서비스(DL Repair Service)를 위한 시스템 설계 및 평가에 적용가능하다.

2. 관련 연구

온톨로지 공학에서 추론 엔진의 추론 결과를 사용자에게 이해하기 쉽게 보여주는 것의 중요성이 점점 높아지고 있다. 추론 결과를 효율적으로 보여주는 것에 대한 첫 번째 시도는 1990년도 CLASSIC[1] 시스템의 개발자에 의해 이루어졌다. 이 시스템 내부의 설명(Explanation) 모듈은 자연적 의미(Natural Semantics) 기반의 귀납적 프레임워크를 사용하여 추론 결과에 대한 정형화된 증명들을 생성했다. 이러한 작업을 위해 해석 모듈은 표현된 서술 논리에 대해 명백히 규정된 증명 규칙(proof rules)이 사용했다.

이와는 다르게 최근에 발표된 테이블로 알고리즘 기반 결정 모듈(Tableau-based Decision Procedure)[2]은 테이블로 알고리즘 기반의 추론 엔진이 실행되는 동안 지식 베이스(Knowledge Base)에 존재하는 공리들 간의 종속 관계를 파악함으로써 추론에 대한 과정을 보여준다. 테이블로 알고리즘 기반 결정 모듈의 주된 목적은 ALC 수준의 서술 논리 TBox에 존재하는 비논리적 개념을 디버깅 하는 것 이다. 본 논문에서 제안하는 종속 부호 기반 비논리적 공리 탐지는 테이블로 알고리즘 기반 결정 모듈을 바탕으로 하고 있으며, SHOIN 수준의 서술 논리까지 처리가 가능하도록 확장되었다. 현재 제안하고자 하는 방법과 가장 연관되는 연구는 Aditya Kalyanpur[3]가 제안한 방법이다. Aditya Kalyanpur가 제안한 방법은 비논리적 개념들로 인해 발생되는 논리적 충돌과 연관된 최소한의 비논리적 공리 집합을 식별한다. 이 방법 역시 테이블로 알고리즘 기반 결정 모듈을 바탕으로 하고 있으며 SHOIN 수준의 서술 논리를 처리할 수 있다.

본 장에서 언급한 기법은 몇 가지 중요한 제약이 있다. 일단 대부분의 시스템은 SHOIN 수준의 서술 논리를 처리할 수 없기 때문에 OWL-DL로 작성된 온톨로지 추론에 적용될 수 없다. *Aditya Kalyanpur가 제안한 기법은 SHOIN 수준의 서술 논리를 처리 할 수는 있지만 비정당성을 유발하는 공리만을 도출하여 표현한 뿐 근원이 되는 비논리적 공리는 식별하지 않는다.* 본 논문에서 제안하고 있는 종속 부호 기반 비논리적 공리 탐지는 이러한 기법들의 제약성을 해결하는 것이 목적이다.

3. 기본 개념

3.1 서술 논리

서술 논리(Description Logics)[4]는 논리 기반 지식 표현(Logic-based Knowledge Representation)의 한 종류이고 1차 술어 논리(First Order Predicate Logic)

의 일부분이다. 따라서 정형화된 형태의 시맨틱 정보 표현이 가능하다. 온톨로지 추론의 기반이 되는 서술 논리(Description Logic)는 기본적으로 개념(Concept), 관계(Role), 개체(Individual)로 사람이 가지고 있는 지식을 표현한다. 개념은 하나의 상수(Constant)가 배치되는 술어(Predicate)로 표현되는데, 이는 온톨로지의 클래스에 해당한다. 관계는 보통 2개의 상수가 배치되는 술어로 표현되며, 이는 온톨로지의 클래스 속성 정의와 개체간의 관계에 해당한다. 개체는 각각의 술어에 포함되는 상수이며, 이는 온톨로지의 개체에 해당한다.

서술 논리는 단일 개념을 조합하여 보다 복합적인 개념을 구성할 수 있도록 구축자(*Constructor*)들을 제공한다. 예를 들어, 다음과 같이 단일 개념인 Person과 Male에 결합.구축자(*Conjunction Constructor* - ⊓)를 적용하여 Male People이라는 복합적인 개념이 정의될 수 있다.

$$\text{Male People} \equiv \text{Person} \sqcap \text{Male}$$

서술 논리로 표현된 지식 베이스(Knowledge Bases)는 일반적으로 *TBox, RBox, ABox*로 구성된다. *TBox*는 일반적인 개념뿐만 아니라 개념 C_1과 C_2에 대해서 $C_1 \sqsubseteq C_2$와 같이 개념의 포함 관계(*Concept Inclusion Axioms*)가 포함되고, *RBox*는 관계 R_1과 R_2에 대해서 $R_1 \sqsubseteq R_2$와 같이 관계의 포함 관계(*Role Inclusion Axioms*)가 포함되며, *ABox*는 개념 *C*와 개체 *a*에 대해서 *C(a)*와 같이 개체의 개념 선언(*Concept Assertions*)과 관계 *R*, 개체 *a, b*에 대해서 *R(a, b)*와 같이 개체의 관계 선언이 포함된다.

현재 서술 논리는 표현 수준에 따라 해당 표현 기호를 통해 분류되어 있다. 그림 1은 표현 기호로 특징화된 서술 논리의 종류를 보여준다. 각 표현 기호는 서술 논리의 특정 표현을 대표한다.

Mnemonic	DL Expressivity
AL	Attribute Logic [A, ¬A (atomic), C ⊓ D, ∃R.⊤ , ∀R.C]
ALC	Attribute Logic + Full Complement [allowing C ⊔ D and ∃R.C]
R⁺	Transitive Roles
S	ALCR⁺
H	Role Hierarchy
I	Inverse Roles
O	Nominals (individuals used in class expressions)
N	Unqualified Cardinality Restriction [≥nR, ≤nR, =nR]
Q	Qualified Cardinality Restriction [(≥ nR).C, (< nR).C, (=nR).C]
D	Datatypes
F	Functional Roles

그림 1 서술 논리의 표현 등급과 기호

3.2 OWL과 온톨로지 추론

웹 온톨로지 언어인 OWL[5]은 시맨틱 웹의 필수적 구성요소인 온톨로지를 구축하는데 사용된다. OWL은 시맨틱 웹 데이타의 생성, 교환 및 공유를 위해 정형화된 사용을 기본으로 한다. 온톨로지 모델링 관점에서 볼 때 OWL은 서술논리가 가지고 있는 많은 논리적 구성과 강하게 일치하고 있다.

OWL은 표현 수준에 따라 OWL-Lite, OWL-DL, OWL-Full 3가지 종류로 나뉜다. 이 중에서 OWL-DL은 SHOIN(D) 수준의 서술 논리와 강하게 일치하고 있다. 따라서 본 논문에서 제안하는 비논리적 공리 탐지 기법은 SHOIN(D) 수준의 서술 논리에 초점이 맞추어져 있다. OWL을 위한 추론 서비스는 일반적으로 서술 논리에 대한 추론서비스와 같이 입력된 온톨로지가 논리적으로 일관성을 갖는지에 대한 일관성 검사, 주어진 온톨로지의 클래스 C와 D 사이에 의미적 포함 관계(*Class Subsumption*)인 $O \models C \sqsubseteq D$가 존재하는지에 대한 검사, 주어진 온톨로지의 개체 a와 클래스 C사이에 개념적 포함 관계(*Instantiation-a*가 *C*의 개체)가 존재하는지에 대한 검사를 한다.

3.3 테이블로 알고리즘

테이블로 기반 알고리즘(*Tableau-based Algorithm*)은 완전 그래프(*Completion Graph*)[6]라고 명명된 모델을 구축하면서 관계 R에 대해서 개념 D의 일관성을 결정한다. 이때 그래프 모델 구축에는 새로운 노드나 간선 추가와 같은 확장된 테이블로 규칙이 적용된다. 예를 들어 x가 개체이고 노드 x의 부호 안에 복합 개념인 C ⊓ D가 존재한다면, x는 C의 개체인 동시에 D의 개체가 된다. 따라서 C, D는 x의 부호에 따로 분리되어 추가 돼야 한다.(테이블로 알고리즘 중에서 ⊓-규칙으로 제어된다.) 다른 예로 만약 개념 ∃R.E가 노드 y의 부호에 존재한다면, y로부터 E를 소속 개념으로 갖는 임의의 개체 노드에 간선 R이 최소한 하나 이상은 존재해야 한다. 그러나 간선이 하나도 존재하지 않는다면 새로운 노드 z를 생성하고, 노드 y로부터 z로 연결되는 간선을 하나 생성한 후, 개념 E를 z의 부호인 L(z)에 추가한다.(테이블로 알고리즘 중에서 ∃-규칙으로 제어된다.)

테이블로 알고리즘은 모델 생성시 논리적 분기 선택을 유발하는 확장 규칙을 고려한다. 예를 들어, 노드의 부호에 개념 C, D가 C⊔D의 논리적 이접(disjunction) 관계로 존재할 때 알고리즘은 C와 D 둘 중 하나를 노드에 추가하게 된다. 논리적 충돌이 발생하게 되면, 테이블로 알고리즘은 논리적 분기의 내용을 바탕으로 나머지 개념을 노드의 부호에 추가한다. 그래프의 모든 말단 노드에 더 이상 적용할 확장 규칙이 없다면 알고리즘은 종료하게 된다. 만약에 종속 부호의 모든 말단 그래프가 논리적 충돌을 포함한다면, '비일관적'(*Inconsistent*)이라는 결과를 내고 개념에 대한 모델이 만들어지지 않는다. 반면에 알고리즘에 의해 생성된 그래프에

어떠한 논리적 충돌도 없다면 추론을 개념에 대해 하나의 가능한 모델을 만들고 '일관적'(*Consistent*)이라는 결과를 낸다.

4. 테이블로 알고리즘 기반 결정 모듈

테이블로 알고리즘 기반 결정 모듈을 이해하기 위해 MUPS(*Minimal Unsatisfiability Preserving Sub-TBoxes*) and 도출과정(*Justification*)에 대한 개념을 이해해야한다. 두 용어에 대한 개념은 다음과 같다.

개념 1. *MUPS(Minimal Unsatisfiability Preserving Sub-TBoxes)*

단일 개념 A에 대한 *MUPS*는 지식 베이스 내에서 A를 정당하게 하지 못하게 하는(*Unsatisfiable*) 최소한의 부분이다. 지식 베이스 K에 대해서 정당하지 못한 개념 C가 주어졌을 때, 만약 K의 일부분인 K'안에서 정당하지 못하면 K'은 C의 *MUPS*가 된다. 이때 C는 $K''\subseteq K'$에 대해서 정당(*Satisfiable*)할 수 있다. *MUPS*는 도출과정(*Justification*)으로 구성된다.

개념 2. 도출 과정(*Justification*)

도출 과정은 하나의 공리가 다른 공리로부터 어떻게 유도되는지 보여준다. 도출 과정은 두 부분으로 구성된다. 후항(*Consequent*)으로 명명되는 도출된 공리와 전항(*Antecedent*)으로 명명되는 도출 원인 공리의 리스트로 구성되며, 본 논문에서는 다음과 같이 도출 과정을 표현 한다.

$$consequent^{[ante1,ante2,...,anten]}$$

5장에서 정의할 비논리적 공리(CIA)는 MUPS의 최소한의 부분이다. 따라서 비논리적 공리와 MUPS는 비슷한 개념을 갖는다. 본 장에서는 MUPS를 탐지하기 위해 기존에 발표되었던 테이블로 알고리즘 기반의 결정 모듈을 소개한다. 이 알고리즘은 SHOIN수준의 서술 논리에서 개념의 정당성을 증명하기 위한 테이블로 알고리즘을 확장한 것이다. 테이블로 알고리즘은 논리적 분기(Non-Deterministic)를 고려한다. 따라서 논리적 충돌이 발생하면 추론 엔진은 또 다른 논리적 분기를 선택하거나, 더 이상 선택할 논리적 분기가 없다면 '비일관적'(*Inconsistent*)이라는 결과를 내고 알고리즘을 종료할 것이다. 그러나 테이블로 알고리즘 기반 결정 모듈의 목적은 입력한 개념에 대해 가능한 모델을 만드는 것이 아니라 입력된 온톨로지 내에서 논리적 충돌을 유발하는 공리들을 식별하는 것이다.

이 알고리즘은 다음과 같이 MUPS를 탐지하기 위해 표준 테이블로 알고리즘 진행에 대해 3가지 변화를 주었다.

1. 이 알고리즘은 노드의 개념을 추가하는 것 외에 논리적 분기점 추가와 논리적 충돌에 원인이 되는 공리 삽입과 같은 명령들을 실행한다.
2. 이 알고리즘은 논리적 충돌이 어떤 논리적 분기 선택에 영향을 받는지에 대해 고려하고, 이를 기반으로 각각의 논리적 충돌 과정을 계산한다. 만약 논리적 충돌이 어떠한 논리적 분기 선택에도 영향을 받지 않으면, 논리적 충돌 과정의 추적 결과물을 직접 추가한다. 만약 논리적 충돌이 논리적 분기 선택에 영향을 받는다면, 이렇게 영향을 받는 모든 논리적 충돌 과정의 추적 결과물을 결합하고, 결합된 최종 결과물을 추가한다.
3. 이 알고리즘은 가능한 모든 논리적 충돌이 표시될 때까지 그래프를 탐색한다.

그래프 G는 다음과 같은 상황에서 논리적 충돌을 포함한다. 노드 x와 개념 C에 대해서 $\{C,\neg C\}\subseteq \mathcal{L}(x)$일 경우와 노드 x, y쌍에 대해서 x와 y가 논리적으로 동등한 이벤트에서 $(x\neq y)$가 존재할 때이다.

테이블로 알고리즘 기반 결정 모듈은 기호 $\mathcal{G}$으로 알고리즘 진행 중 발생하는 모든 이벤트(논리적 충돌)를 저장한다. 이 알고리즘은 그래프의 변화를 기록하기 위해서, 이벤트가 일어났을 경우 발생하는 논리적 분기 선택과 공리들을 종속적인 순서대로 기록하는 과정 탐지 함수(*Tracing Function*)를 적용한다. 과정 탐지 함수 τ는 각 이벤트의 해당 집합으로 매핑 되는데, 각 집합은 논리적 분기 지점 α와 공리들을 포함한다. 따라서 모델 구축을 무의미화 시키는 모든 논리적 충돌에 대해 기록하며, 더 이상 논리적 충돌이 발견되지 않거나 더 이상 고려할 논리적 분기 선택이 남아있지 않을 때까지 백트래킹을 계속한다. 만약 논리적 충돌이 논리적 분기 선택에 영향을 받지 않으면 바로 *MUPS*를 계산한다. 그러나 논리적 충돌이 논리적 분기 선택에 영향을 받는다면 논리적 분기점의 또 다른 논리적 분기를 고려할 필요가 있다. 알고리즘 진행 과정에서 더 이상 적용할 규칙이 없을 때는 현재의 논리적 분기를 삭제하고 가장 최근에 방문한 논리적 분기점 α로 백 트래킹한 후, 다른 논리적 분기선택을 적용하여 그래프를 확장해나간다. 만약 α에 더 이상 고려해야할 논리적 분기가 없다면, α이전부터 가장 최근에 방문한 α'으로 백트래킹 한다.

테이블로 알고리즘은 *SHOIN* 수준에 서술 논리의 개념 정당성을 증명하기 위해 *2NEXPTIME*[7]의 실행시간을 소모한다. 따라서 본 장에서 설명한 테이블로 알고리즘 기반의 결정 모듈은 *2NEXPTIME*의 복잡도를 갖는다.

5. 종속 부호 기반 비논리적 공리 탐지

본 논문에서 제안하는 기법인 종속 부호 기반 비논리적 공리 탐지를 이해하기 위해서는 비논리적 공리(CIA)

와 공리의 가정(*Assumptions*) 및 종속 부호(*Dependency Label*)의 개념을 이해할 필요가 있다. 세 용어에 대한 개념은 다음과 같다.

개념 3. CIA(*Causing Inconsistency Axiom*)

단일 개념 A에 대한 CIA는 A를 정당하게 하지 못하게 하는 MUPS의 최소한의 부분이다. 지식 베이스 K에 대해서 정당하지 못한 개념 C가 주어졌을 때, 만약 K의 일부분인 K'안에서 정당하지 못하면 K'은 C의 CIA가 된다. 그리고 항상 C는 $K'' \subseteq K'$에 대해서 정당하지 못하다. CIA를 탐지하기 위해서는 도출과정 뿐 아니라 공리의 가정(Assumptions)도 고려해야 한다.

개념 4. 공리의 가정(*Assumptions*)

공리의 가정은 특정 공리를 유도하는 단일 공리를 나타낸다. 따라서 공리의 가정은 도출 과정의 전항으로부터 계산되어진다.

개념 5. 종속 부호(*Dependency Label*)

종속 부호는 공리의 가정들과 도출과정으로 구성된 집합이다. 따라서 종속 부호는 CIA를 찾기 위한 기본 데이타를 제공한다. 종속 부호의 데이타 구조는 그래프 노드 형태이며, 본 논문에서는 다음과 같이 표현한다.

$$Axiom^{[\{a\ set\ of\ assumption\},\{a\ set\ of\ justification\}]}$$

테이블로 알고리즘 기반의 결정 모듈은 오직 비논리적 공리들의 집합을 최소화하고, 실행시간 역시 많이 소모한다(2NEXTTIME). 본 장에서는 예제를 통해 제안하는 기법의 핵심 개념을 설명한 후, 알고리즘에 대한 설명을 한다. 본 논문에서는 서술 논리 추론을 위한 테이블로 기반 추론 알고리즘에 대해 자세한 설명은 생략한다.

본 논문에서 제안하는 CIA 탐지 전략은 그래프와 부호의 내용이 변화할 때마다 공리의 가정(Assumptions)과 도출과정(*Justification*)을 구성하는 것이다. 따라서 종속 부호 기반 비논리적 공리 탐지를 위한 알고리즘은 세 단계로 진행된다.

1. 비논리적 공리 탐지 과정에서 그래프 구축 중 실행되는 다양한 명령어들(부호에 개념 추가, 논리적 분기 공리 삽입, 논리적 충돌 지점 삽입)을 사용하여 공리의 가정과 도출 과정을 유지한다.
2. 가장 먼저 탐지된 최소화된 비논리적 공리 집합과 비논리적 공리 유발에 원인을 제공 하는 공리를 저장하고, 이들 중 근원 공리와 논리적 충돌에 영향을 주는 논리적 분기 개념을 지식 베이스에서 삭제한다.
3. 논리적 충돌 없는 그래프가 구성될 때까지 비논리적 공리 탐지 과정을 반복한다.

최소화된 비논리적 공리 집합(CIA)을 탐지하는 기법은 다음과 같다.

MUPS ← 논리적 충돌을 발생 시키는 개념1의 가정
∪ 논리적 충돌을 발생 시키는 개념2의 가정

비논리적 공리 유발의 원인을 제공하는 공리 후보 ← 논리적 충돌을 발생 시키는 개념1의 가정
∩ 논리적 충돌을 발생 시키는 개념2의 가정

비논리적 공리 유발의 원인을 제공하는 공리 ← 비논리적 공리 유발의 원인을 제공하는 공리 후보 리스트 중 마지막에 위치한 개념

최소화된 비논리적 공리 집합(CIA) ← MUPS를 구성하는 리스트 중 비논리적 공리 유발의 원인을 제공하는 공리와 그 다음에 위치하는 나머지 공리들

5.1 예제

본 절에서는 먼저 간단한 예제를 통해 핵심 개념을 설명하고, 이를 확장하여 좀 더 복잡한 상황을 설명한다. 먼저 지식 베이스(Knowledge Base-KB)에 다음과 같은 공리들이 존재한다고 가정한다.

예제 1.

1. $Hamburger \sqsubseteq Fast\_Food \sqcap \exists kindOf.Hot\_Food$
2. $Hamburger \sqsubseteq Good\_Food \sqcup \forall kindOf.Fast\_Food$
3. $Fast\_Food \sqsubseteq QuickMakeFood \sqcap CheapishFood$
4. $QuickMakeFood \sqsubseteq (\geq 1.hasIngredient)$

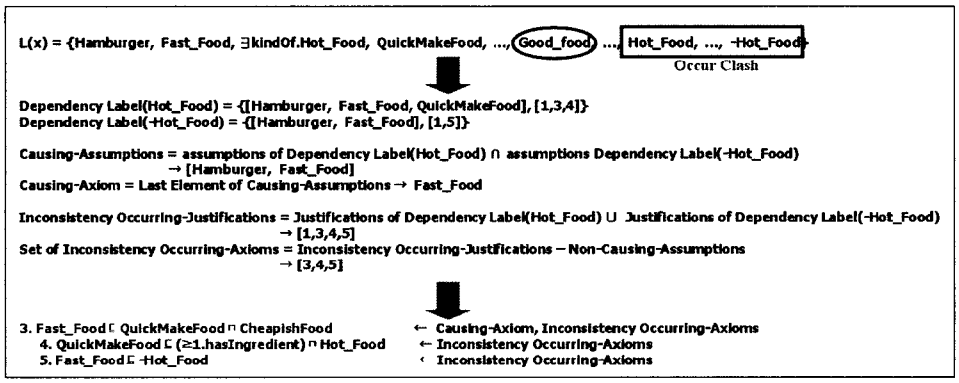

그림 2 비논리적 공리 유발에 근원이 되는 공리 결정에 관한 예제

$\sqcap Hot\_Food$

5. $Fast\_Food \sqsubseteq \neg Hot\_Food$

예제 1에서 현재 지식 베이스에 존재하는 비논리적인 개념은 *Hamburger*라는 사실과 비논리적 개념을 유발하는 공리들 중 근원 공리(Causing-Axiom)는 $Fast_Food \sqsubseteq \neg Hot_Food$라는 것을 알 수 있다. 또한 *Hamburger*에 대해서 최소화된 비논리적 공리 집합은 {3,4,5}이다. 그림 2는 *Hamburger*에 테이블로 알고리즘을 적용하므로 써 생성된 1개의 노드로 구성된 그래프를 보여준다. 이 그래프는 *Hamburger*에 대한 하나의 단일 노드와 *Hamburger*를 포함하고 있는 부호(Label)로 초기화되며, 테이블로 알고리즘의 확장 규칙에 의해서 그래프와 부호의 내용이 증가한다.

그림 2에서 보듯이 공리의 가정과 도출 과정은 각 개념의 종속 부호(Dependancy Label)에 명시된다. 본 예제에서 특별히 주목되는 부분은 2번 공리처럼 논리적인 분기 선택(*Non-Deterministic Choices*)이 생긴다는 것이다. 그러나 예제 1에서는 논리적인 분기 선택에 상관없이 같은 논리적 충돌(Clash)이 발생한다. 따라서 논리적인 분기 선택은 고려하지 않는다.

*Hamburger*에 대한 MUPS는 {1,3,4,5}이다. 그러나 MUPS는 논리적 충돌을 유발하는 공리와 관계있는 공리들의 집합이므로, MUPS의 모든 내용이 비논리적 공리 탐지에 필요하지는 않다. MUPS는 논리적 충돌을 발생시키는 두 개념의 가정을 논리합(Union)으로 연결하여 구한다. 반면에 비논리적 공리 유발에 원인을 제공하는 근원 공리(Causing-Axiom)는 논리적 충돌을 발생시키는 두 개념의 가정을 논리곱(Intersection)으로 연결한 후 결과 리스트의 가장 끝에 위치한 개념을 선택한다. 그리고 MUPS를 구성하는 리스트 중 비논리적 공리 유발에 원인을 제공하는 공리와 그 다음에 위치하는 나머지 공리들이 CIA가 된다. 따라서 비논리적 공리 탐지 과정에 생성된 공리의 가정과 도출 과정을 통해 *Fast_Food*가 비논리적 공리 유발에 근원이 되는 공리(*Causing-Axiom*)를 결정할 수 있으며 *Fast_Food*에 영향을 받는 MUPS의 부분집합인 {3,4,5}가 최소화된 비논리적 공리 집합(CIA)으로 결정된다.

예제 2.

6. $Fast\_Food \sqsubseteq \neg Good\_Food$

예제 2는 좀 더 복잡한 상황을 보여준다. 지식 베이스에 *Fast_Food*에 대한 새로운 공리가 추가되었다. 그림 3은 새롭게 추가된 공리에 대한 그래프를 보여준다. 그림의 첫 번째 부분에 나타나듯이 논리적인 분기 선택으로 *Good_Food*를 선택할 경우 논리적 충돌 c1이 발생한다. 따라서 예제 2는 논리적인 분기 선택에 따라 논리적 충돌이 발생한다는 것을 보여준다.

이와 같은 상황이 발생할 경우, 제안한 방법은 논리적 충돌 c1을 일으키는 논리적 분기 공리와 c1의 근원이

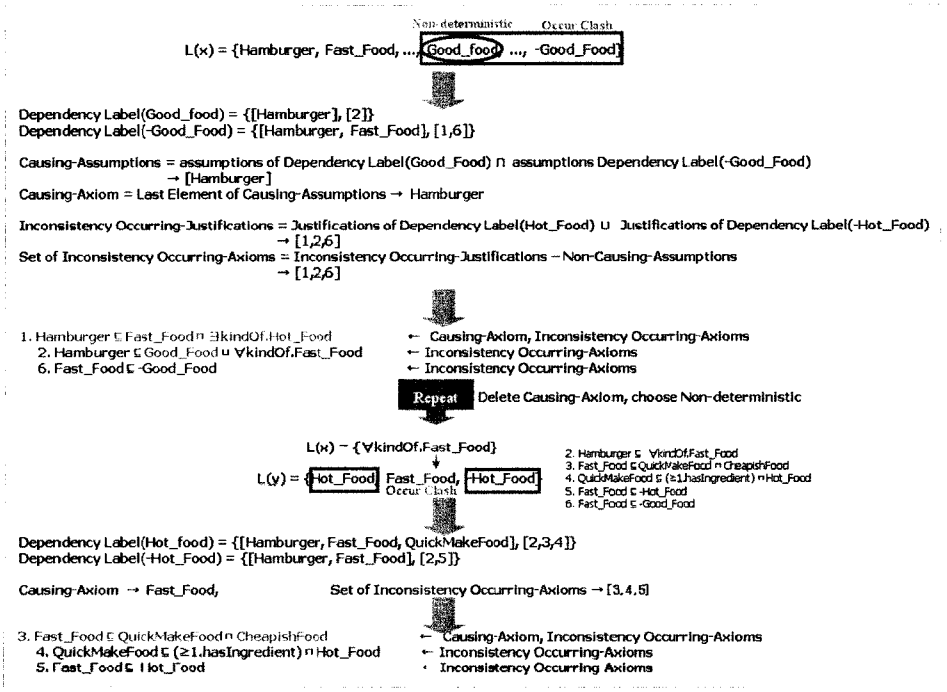

그림 3 비논리적 공리 유발에 근원이 되는 공리 결정에 관한 예제

되는 공리(Causing-Axiom)를 지식 베이스에서 삭제한 후 비논리적 공리 탐지 과정을 반복 실행한다. 그림의 두 번째 부분에 나타나듯이 *Hot_Food*와 ¬*Hot_Food*를 포함하고 있는 부호 *L(y)* 에서 논리적 충돌 c2가 발생한다. 따라서 c2의 근원이 되는 공리가 추가로 탐지되며, 최종적으로 c1과 c2를 발생시키는 비논리적 공리와 근원이 되는 공리가 온톨로지 사용자 및 설계자에게 보이게 된다. 마지막 예제에서 다음과 같은 공리가 지식 베이스에 추가는 것을 고려한다.

예제 3.

7. Hamburger ⊑ (QuickMakeFood ⊓ -QuickMakeFood) ⊔ -Good_Food

그림 4에서 보듯이 추가되는 공리는 새로운 논리적 분기 선택과 논리적 충돌을 발생시킨다. 논리적 충돌 c1은 부호 *L(y)*의 *QuickMakeFood*와 ¬*QuickMakeFood* 사이에서 발생한다. 논리적 충돌 c1의 발생은 첫 번째 논리적 분기 선택인 *Good_Food*에 영향을 받지 않지만, 두 번째 논리적 분기 선택인 *(QuickMakeFood ⊓ -QuickMakeFood)*에 영향을 받는다. 두 번째 비논리적 공리 탐지 과정에서 논리적 충돌 c2는 첫 번째 논리적 분기 선택인 *Good_Food*에 의해서 발생된다. 따라서 그림 4에서 보듯이 3번의 탐지 과정을 반복한 후 비논리적 근원이 되는 공리 *Hamburger, Fast_food*와 이에 해당하는 CIA가 탐지된다.

5.2 종속 부호 기반 비논리적 공리 탐지

본 절에서는 비논리적 공리를 탐지하기 위해 보다 확장시킨 테이블로 알고리즘을 정의한다. 본 알고리즘은 완전 그래프 구축 외에도 입력된 서술 논리의 지식 베이스에 대한 공리의 가정과 도출과정을 종속 부호에 저장한다.

이 알고리즘의 확장된 규칙을 적용함으로써 알고리즘의 흐름이나 완전 그래프의 상태 변화에 따라 생성되는 공리의 가정과 도출과정들이 기록된다. 완전 그래프에서의 변화를 기록하기 위해서, 두 개의 기록 함수인 ĵ(도출과정 기록), ă(가정 기록)와 상등 관계 카운터(*Equality Counter*) ń를 추가했다. 표 1은 종속 부호 기반 비논리적 공리 탐지를 위해 확장한 테이블로 알고리즘을 보여주며, 표 2는 이 알고리즘이 실행되는 과정을 정리한 것이다.

6. 실험 및 평가

본 장에서는 비논리적 공리를 탐지하기 위해 제안한 기법에 대한 성능을 평가한다. 평가를 위해 기존에 발표되었던 테이블로 알고리즘 기반의 결정 모듈과 성능 비

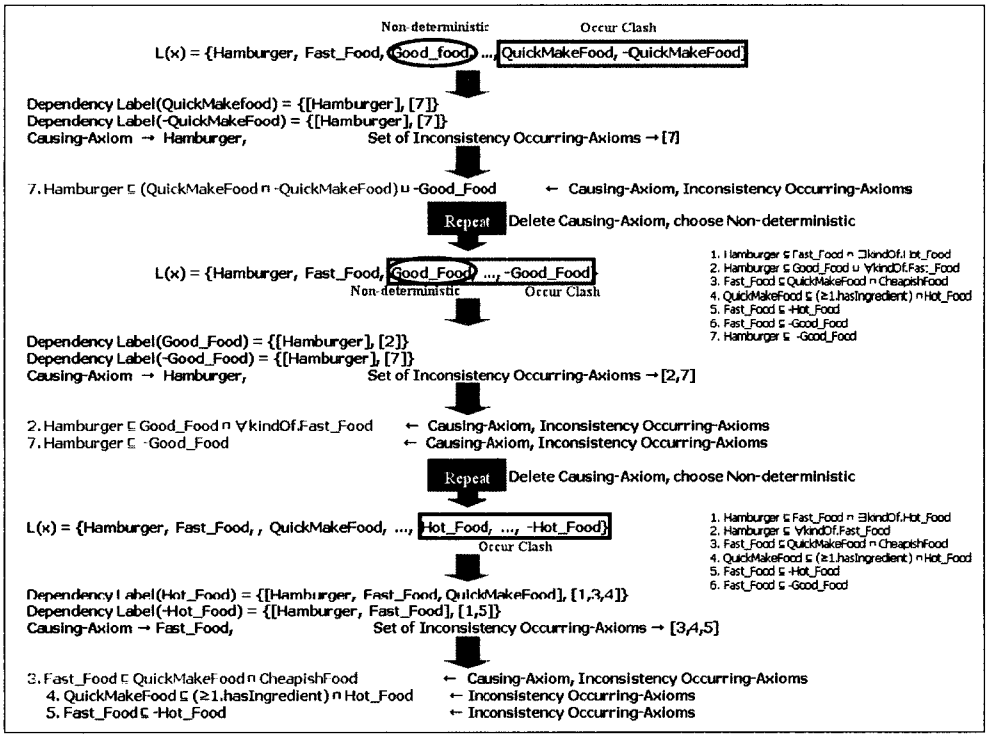

그림 4 비논리적 공리 유발에 근원이 되는 공리 결정에 관한 예제

표 1 종속 부호 기반 비논리적 공리 탐지를 위해 확장된 테이블로 알고리즘

$\sqcap$ rule : $(C_1 \sqcap C_2) \in \mathcal{L}(x)$, x is indirectly blocked, then if $(C_1, C_2) \not\subseteq \mathcal{L}(x)$, then Assert($\mathcal{L}(x)$, $\{C_1, C_2\}$) $\mathfrak{j}(C_1) \leftarrow \mathfrak{j}(C_1) \otimes \mathfrak{j}(C_1 \sqcap C_2)$, $\mathfrak{j}(C_2) \leftarrow \mathfrak{j}(C_2) \otimes \mathfrak{j}(C_1 \sqcap C_2)$ $\breve{a}(C_1) \leftarrow \breve{a}(C_1) \otimes \breve{a}(C_1 \sqcap C_2)$, $\breve{a}(C_2) \leftarrow \breve{a}(C_2) \otimes \breve{a}(C_1 \sqcap C_2)$
$\sqcup$ rule : $(C_1 \sqcup C_2) \in \mathcal{L}(x)$, x is indirectly blocked, then if $(C_1, C_2) \not\subseteq \mathcal{L}(x)$, then select C_i, $i \in \{1,2\}$, Assert($\mathcal{L}(x)$, C_i) Division($C_1 \sqcup C_2$) $\mathfrak{j}(C_i) \leftarrow \mathfrak{j}(C_i) \otimes \mathfrak{j}(C_1 \sqcup C_2)$, $\breve{a}(C_1) \leftarrow \breve{a}(C_1) \otimes \breve{a}(C_1 \sqcup C_2)$
$\exists$ rule : $\exists S.C \in \mathcal{L}(x)$, x is indirectly blocked, then if x has no S-neighbor y with $C \in \mathcal{L}(x)$, then create new node y, Assert($\mathcal{L}(x,y)$,S), Assert($\mathcal{L}(y)$, C) $\mathfrak{j}(C) \leftarrow \mathfrak{j}(C) \otimes \mathfrak{j}(\exists S.C)$, $\mathfrak{j}(S) \leftarrow \mathfrak{j}(S) \otimes \mathfrak{j}(\exists S.C)$ $\breve{a}(C) \leftarrow \breve{a}(C) \otimes \breve{a}(\exists S.C)$, $\breve{a}(S) \leftarrow \breve{a}(S) \otimes \breve{a}(\exists S.C)$
$\forall$ rule : $\forall S.C \in \mathcal{L}(x)$, x is indirectly blocked, and there is an R-neighbor y of x, then if $C \notin \mathcal{L}(y)$, then Assert($\mathcal{L}(y)$, C) $\mathfrak{j}(C) \leftarrow \mathfrak{j}(C) \otimes \mathfrak{j}(\forall S.C)$, $\breve{a}(C) \leftarrow \breve{a}(C) \otimes \breve{a}(\forall S.C)$
$\forall^+$ rule : $\forall S.C \in \mathcal{L}(x)$, x is indirectly blocked, and there is an R-neighbor y of x, then Trans(R) and $R \sqsubseteq S$, then if $\forall S.C \notin \mathcal{L}(y)$, then Assert($\mathcal{L}(y)$, $\forall S.C$) $\mathfrak{j}(\forall S.C, y) \leftarrow \mathfrak{j}(\forall S.C, y) \otimes \mathfrak{j}(\forall S.C, x)$ $\breve{a}(\forall S.C, y) \leftarrow \breve{a}(\forall S.C, y) \otimes \breve{a}(\forall S.C, x)$
$\geq$ rule : $(\geq nS.C) \in \mathcal{L}(x)$, x is indirectly blocked, then if there is no n safe S-neighbors $y_1,...,y_n$ of x with $y_i \neq y_j$ then create n new node $y_1,...,y_n$, Assert($\mathcal{L}(y_i)$, C) $\mathfrak{j}(C, y_i) \leftarrow \mathfrak{j}(C, y_i) \otimes \mathfrak{j}(\geq nS.C, x)$, $\breve{a}(C, y_i) \leftarrow \breve{a}(C, y_i) \otimes \breve{a}(\geq nS.C, x)$
$\leq$ rule : $(\leq nS.C) \in \mathcal{L}(x)$, x is indirectly blocked, and there are m $y_1,...,y_m$ of x with $m > n$, then select two S-neighbors of x: y, z if z is nominal node $\rightarrow$ Merge(y,z) else if y is a nominal node, or ancestor of z $\rightarrow$ Merge(z,y) else Merge(y,z) Select(y, z), Equ(y=z) $\mathfrak{j}(C, y=z) \leftarrow \mathfrak{j}(C, y=z) \otimes \mathfrak{j}(C, y) \otimes \mathfrak{j}(C, z)$ $\breve{a}(y=z) \leftarrow \breve{a}(y=z) \otimes \breve{a}(C, y) \otimes \breve{a}(C, z)$ $\dot{\eta}(x) \leftarrow \cup \mathcal{L}\langle x, y_n \rangle$
O rule : if, for some $\{o\} \in N_I$ there are 2 nodes x,y with $\{o\} \in \mathcal{L}(x) \sqcap \mathcal{L}(y)$, not $x \neq y$, then Merge(x,y) Equ(x,y) $\mathfrak{j}(\{o\}, x=y) \leftarrow \mathfrak{j}(\{o\}, x=y) \otimes \mathfrak{j}(\{o\}, x) \otimes \mathfrak{j}(\{o\}, y)$ $\breve{a}(\{o\}, x=y) \leftarrow \breve{a}(\{o\}, x=y) \otimes \breve{a}(\{o\}, x) \otimes \breve{a}(\{o\}, y)$ $\dot{\eta}(x) \leftarrow$ Nequ(x,y)
NN rule : if, $(\leq nS.C) \in \mathcal{L}(x)$, x is a nominal node, and there is a blockable S-neighbor y of x and x is a successor of y, there is no m such that $1 \leq m \leq n$, $(\leq mS.C) \in \mathcal{L}(x)$, and there exist m nominal S-neighbors $z_1,...,z_m$ of x. $z_i \neq z_j$, $1 \leq i \leq j \leq m$, then guess m with $1 \leq m \leq n$. Assert($\mathcal{L}(x)$, $\leq mS.C$), create m new nodes $y_1,...,y_m$. Assert($\mathcal{L}(\langle x,y_i \rangle)$,S),Assert($\mathcal{L}(y_i).\{C,o_i\}$), for each $o_i \in N_I$ new in G and $y_i \neq y_j$ for $1 \leq i \leq j \leq m$. $\mathfrak{j}(C, y_i) \leftarrow \mathfrak{j}(C, y_i) \otimes \mathfrak{j}(\leq nS.C, x)$, $\breve{a}(C, y_i) \leftarrow \breve{a}(C, y_i) \otimes \breve{a}(\leq nS.C, x)$

교를 하였다. 실험에 사용한 온톨로지는 *Pizza Ontology*[8]와 Swoop 프로젝트[9]에서 사용한 온톨로지들이며, 이 중 Pizza 온톨로지의 경우 인위적으로 논리적 충돌에 직접적인 원인을 제공하는 192개의 비논리적 공리를 추가하였다. 이렇게 비논리적 공리를 생성함으로써 본 논문에서 제안하는 기법의 결과물을 확인하고, 비논

표 2 확장된 테이블로 알고리즘의 실행 과정

```
Running Extend Tableaux Algorithm
  if clash-event ← ⊓ rule, ⊔ rule, ∃ rule, ∀ rule, ∀+ rule, ≥ rule
      stream-out ← ʃ and ă
  else clash-event ← ≤ rule, O rule
      stream-out ← ṅ(x)
```

표 3 실험에 사용된 온톨로지 정보

	비논리적 공리	전체 공리
Pizza Ontology	191	323
bad-food	3	33
buggyPolicy	6	57
buggy-sweet-jpl	0	3321
koala	2	28
madcow	1	75
miniTambis	37	254
oldUniversity	6	37
univ-bench	0	51
University	11	39

리적 공리 탐지에 대한 정밀도(*Precision*)와 재현율(*Recall*) 및 비논리적 공리를 탐지하는데 소모되는 시간을 측정하였다.

본 논문에서 제안한 기법의 최종 목적은 정확하게 논리적 충돌 원인을 제공하는 공리를 탐지하는 것이다. 따라서 성능 평가 부분에서는 어느 정도 온톨로지의 완성도에 영향을 받게 된다.

Pizza 온톨로지를 기준으로 제안한 기법과 테이블로 알고리즘 기반 결정 모듈을 비교했을 때, 제안한 방법을 통해 탐지된 191개의 공리(논리적 충돌 원인을 제공하는 공리들) 중 오직 2개의 공리만이 부정확(논리적 충돌 원인과 연관이 없는 공리)한 반면, 테이블로 알고리즘 기반 결정 모듈을 통해 탐지된 205개의 공리들 중 83개의 공리가 부정확하였다. 표 4는 두 기법의 정밀도와 재현율을 정리한 것이다.

표 5는 Swoop 프로젝트에서 사용한 온톨로지를 기준으로 비교실험을 한 결과이다. buggy-sweet-jpl 온톨로지와 univ-bench 온톨로지는 비일관성을 유발하는

표 4 Pizza 온톨로지에 대한 비논리적 공리 탐지의 정밀도와 재현율

	종속 부호 기반 비논리적 공리 탐지 기법	테이블로 알고리즘 기반 결정 모듈
정확하게 탐지된 비논리적 공리	189	122
탐지된 공리	191	205
정밀도	0.99	0.60
재현율	0.98	0.64

공리가 없기 때문에 실험에서 제외하였다. Swoop 프로젝트에서 사용한 온톨로지의 경우 비일관성을 유발하는 공리가 상대적으로 적었기 때문에 MUPS나 CIA를 구성하는 비논리적 공리의 수도 역시 적다. 이들 온톨로지의 실험 결과를 종합하여 평가하면 제안하는 방법이 정밀도와 재현율 모두 높다는 것을 알 수 있다.

따라서 종속 부호 기반 비논리적 공리 탐지 기법이 좀 더 비논리적 공리를 정확하게 찾아내는 것을 알 수 있다. *그러나 시간 소모율을 비교하면은 테이블로 알고리즘 결정 모듈에 비해서 온톨로지의 전체적인 공리 개수에 비례하여 일반적으로 (0.02ms/공리 개수)배의 시간을 더 소모한다.* 따라서 정확성을 유지하면서 소모 시간을 줄이는 방법이 필요하다.

7. 결 론

대부분의 추론엔진들은 OWL 온톨로지로부터 정당하지 못한 개념의 탐지 과정 없이 단순히 추론된 결과만을 보여준다. 본 논문에서는 온톨로지 디버깅을 위한 종속 부호 기반 비논리적 공리 탐지 기법에 대해 제안했다. 비논리적 공리를 탐지하기 위해서, 온톨로지 내에서

표 5 Swoop 프로젝트에 사용된 온톨로지에 대한 비논리적 공리 탐지의 정밀도와 재현율

Swoop 온톨로지	종속 부호 기반 비논리적 공리 탐지 기법			테이블로 알고리즘 기반 결정 모듈		
	정밀도	재현율	속도	정밀도	재현율	속도
bad-food	1.00	1.00	24ms	0.66	0.66	22ms
buggyPolicy	1.00	0.83	20ms	0.89	0.82	21ms
koala	1.00	1.00	16ms	1.00	1.00	17ms
madcow	1.00	1.00	48ms	0.00	0.00	36ms
miniTambis	0.97	0.94	97ms	0.71	0.52	78ms
oldUniversity	1.00	1.00	19ms	0.88	0.74	19ms
University	0.99	0.99	37ms	0.83	0.67	31ms

비논리적 공리를 유발하는 근원 공리를 찾아야 될 필요가 있다. 본 논문에서 제안한 종속 부호 기반 비논리적 공리 탐지 기법을 기존의 테이블로 알고리즘 기반의 결정 모듈과 비교했을 때, 비논리적 공리를 찾는데 있어서 최대 34% 가량의 좋은 성능을 보였다. 그러나 온톨로지의 비논리적 공리 개수에 비례하여 (0.02ms/공리 개수) 배의 시간을 더 소모하였다. 따라서 향후 비논리적 공리를 탐지하는데 있어서, 정확성을 유지하면서 소모 시간을 줄이는 방법에 대한 연구가 진행될 것이다.

참 고 문 헌

[1] McGuinness, D. and Borgida, A. Explaining Subsumption in Description Logics. Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence.

[2] Schlobach, S. and Cornet, R. Non-Standard Reasoning Services for the Debugging of Description Logic Terminologies Proceedings of IJCAI, 2003.

[3] Aditya Kalyanpur, Bijan Parsia, Bernardo Cuenca Grau and Evren Sirin. Justifications for Entailments in Expressive Description Logics. Technical report.

[4] F. Baader andW. Nutt. Basic description logics. In Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, and Peter F. Patel-Schneider, editors, *The Description Logic Handbook: Theory, Implementation, and Applications*, pages 43-95. Cambridge University Press, 2003.

[5] M. Dean and G. Schreiber. OWL Web Ontology Language Reference W3C Recommendation. http://www.w3.org/tr/owl-ref/. February 2004.

[6] Ian Horrocks and Ulrike Sattler. A tableaux decision procedure for SHOIQ. In *Proc. of IJCAI 2005*, 2005.

[7] S. Tobies Complexity Results and Practical Algorithms for Logics in Knowledge Representation PhD Dissertation 2001.

[8] Nick Drummond, Matthew Horridge, Robert Stevens, Chris Wroe, Sandra Sampaio http://www.co-ode.org/ontologies/pizza/2007/02/12/, 2007.

[9] Debugging OWL Ontologies using Swoop, http://www.mindswap.org/2005/debugging/, 2005.

김 제 민

2001년 숭실대학교 컴퓨터학과(학사). 2004년 숭실대학교 대학원 컴퓨터학과(석사) 2004년~현재 숭실대학교 대학원 컴퓨터학과 박사과정. 관심분야는 인공지능, 시맨틱 웹, 유비쿼터스 컴퓨팅

박 영 택

1978년 서울대학교 전자공학과(학사). 1980년 KAIST 전산학(석사). 1992년 Univ. of Illinois at Urbana Champaign(박사). 1981년~현재 숭실대학교 컴퓨터학과 교수. 관심분야는 인공지능, 에이전트, 전문가 시스템, 시멘틱 웹