NATMS를 이용한 온톨로지 추론의 non-deterministic 문제 해결 및 일관성 오류 탐지 기법

(Solving Non-deterministic Problem of Ontology Reasoning and Identifying Causes of Inconsistent Ontology using Negated Assumption-based Truth Maintenance System)

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

요 약 온톨로지의 논리적 오류와 개념들 간의 포함 관계를 탐지하는 추론 엔진들이 소개되고 있다. 발표된 온톨로지 추론 엔진의 대부분은 태블로 알고리즘을 기반으로 구축되었다. 그러나 대부분의 추론 엔진들은 논리적 오류를 일으키는 원인은 밝히지 않고, 논리적 오류를 갖는 개념만을 탐지한다. 본 논문의 목적은 태블로 알고리즘 전개 과정 중에 발생하는 non-deterministic 상황을 최적화 하는 동시에 논리적 오류를 일으키는 원인을 탐지하기 위한 방법을 연구하는 것이다. 따라서 본 논문에서는 논리적 부정 가정 기반 진리 유지 시스템(NATMS)을 사용하여 non-deterministic 문제를 해결하고 논리적 오류 원인을 탐지하는 기법을 제안한다. 본 논문에서는 기존에 발표되었던 종속 부호 기반 백트랙킹 기법과 Swoop 프로젝트에 적용된 논리적 오류 원인을 탐지하는 기법을 소개하고, 제안하고자 하는 기법을 설명한다.

키워드 : 온톨로지, 추론 엔진, 최적화 기법, 종속 부호 기반 백트랙킹, 가정 기반 진리 유지 시스템

Abstract In order to derive hidden information (concept subsumption, concept satisfiability and realization) of OWL ontology, a number of OWL reasoners have been introduced. The most of these ontology reasoners were implemented using the tableau algorithm. However most reasoners simply report this information without providing a justification for any arbitrary entailment and unsatisfiable concept derived from OWL ontologies. The purpose of this paper is to investigate an optimized method for non-deterministic rule of the tableau algorithm and finding axioms to cause inconsistency in ontology. In this paper, therefore, we propose an optimized method for non-deterministic rule and finding axiom to cause inconsistency using NATMS. In the first place, we introduce Dependency Directed Backtracking to deal non-deterministic rule, a tableau-based decision procedure to find unsatisfiable axiom Furthermore we propose an improved method adapting NATMS.

Key words : Ontology, Ontology Reasoner, Optimized Method, Dependancy Directed Backtracking, Assumption based Truth Maintenance System

· 본 논문은 숭실대학교의 지원을 받았습니다.

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

논문접수 : 2008년 10월 29일
심사완료 : 2009년 3월 8일

정보과학회논문지 : 소프트웨어 및 응용 제36권 제5호(2009.5)

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. 현재 온톨로지가 점점 대용량화되면서 구축 과정이 점점 복잡해지고 있다. 대형 온톨로지를 구성할

경우 서술 논리 기반의 지식 표현에 익숙지 않거나 전문가가 아니라면, 논리적 오류 없이 온톨로지를 구축하는 것은 매우 어려운 작업이 된다. 따라서 구축 중인 온톨로지의 정확한 논리적 오류를 찾아내고, 이를 수정하는 작업은 매우 중요하다고 할 수 있다. 또한 온톨로지를 온톨로지 기반의 검색에 적용하거나, 다른 온톨로지와 통합할 때, 온톨로지를 구성하는 개념들 간의 포함 관계를 추론하는 것 역시 중요하다.

현재 온톨로지의 논리적 오류와 개념들 간의 포함 관계를 탐지하는 추론 엔진들이 소개되고 있다. 이 중 FaCT, FaCT++[1]가 많이 이용되고 있으며, 최근에는 Pellet[2]과 KAON2[3]의 활용도 점차 증가되고 있는 추세다. 온톨로지 추론 엔진을 구축하는 연구는 서술 논리로 표현된 온톨로지의 정확하면서 완전한 추론을 합리적인 시간 안에 수행할 수 있는 알고리즘 개발에 집중되고 있다. 온톨로지 추론 엔진의 대부분은 태블로 알고리즘을 기반으로 구축되었다. 태블로 알고리즘 기반의 온톨로지 추론 엔진은 규칙 기반 온톨로지 추론 엔진에 비해서 정확하면서 완전한 결과를 보여준다. 그러나 태블로 알고리즘은 non-deterministic 처리에 있어서 시간과 공간 복잡도가 높기 때문에, 이를 해결해 줄 최적화 기법이 필요하다. 또한 대부분의 태블로 알고리즘 기반의 추론 엔진들은 논리적 오류를 일으키는 원인은 밝히지 않고, 논리적 오류를 갖는 개념만을 탐지한다.

본 논문의 목적은 태블로 알고리즘 전개 과정 중에 발생하는 non-deterministic 상황을 최적화 시키는 동시에 논리적 오류를 일으키는 원인을 탐지하기 위한 방법을 연구하는 것이다. 따라서 본 논문에서는 논리적 부정 가정 기반 진리 유지 시스템(Negated Assumption based Truth Maintenance System-NATMS)을 사용하여 non-deterministic 문제를 해결 하고 논리적 오류 원인을 탐지하는 기법을 제안한다. 가정 기반 진리 유지 시스템은 추론 엔진의 추론 과정을 기억하고 각 추론 상태의 진위를 관리하는 기능을 수행한다. 이는 ATMS를 구성하는 노드 레이블과 노굿(Nogood)을 관리함으로써 추론 과정에서 논리적 모순이 발생하였을 때 이를 효과적으로 처리하는 것이다.

현재, 태블로 알고리즘의 non-deterministic 문제 해결을 위한 최적화 기법으로 종속 부호 기반 백트랙킹(Dependency Directed Backtracking-DDB)[4]이 발표되었다. 종속 부호 기반 백트랙킹은 non-deterministic 한 상황이 발생 할 때마다 분기점(Branch Point)을 종속 부호(Dependency Label)에 기록함으로써, 논리적 모순이 발생하였을 경우, 논리적 모순의 직접적인 원인을 제공하는 분기점으로 직접 백트랙킹 하는 기법이다. 논리적 오류 원인탐지와 관련된 연구로는 Swoop 프로젝트에서 사용된 종속 부호 기반 MUPS(Minimal Unsatisfiability Preserving Sub-TBoxes)구성 기법[5]이 있다. MUPS는 논리적 오류에 관계되는 공리(Axiom)들의 집합인데, 이러한 집합들을 구성함으로써 수정될 온톨로지의 부분을 최소화시켜 정확하게 표현한다. 이 기법은 태블로 규칙이 적용될 때마다 레이블에 추가되는 공리들의 종속 부호에 자신을 도출하는 공리들을 종속적인 순서대로 기록한다.

이에 본 논문에서는 먼저 기존에 발표되었던 종속 부호 기반 백트랙킹 기법과 Swoop 프로젝트에 적용된 종속 부호 기반 MUPS 구성 기법을 소개하고, 제안하고자 하는 기법을 설명한 후 비교 실험을 통해 성능을 측정한다.

2. 관련 연구

온톨로지 공학에서 추론 엔진의 추론 결과를 사용자에게 이해하기 쉽게 보여주는 것에 대한 중요성이 점점 높아지고 있다. 추론 결과를 효율적으로 보여주는 것에 대한 첫 번째 시도는 1990년도 CLASSIC[6] 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.

최근에 발표된 태블로 알고리즘 기반 결정 모듈(Tableau based Decision Procedure)[7]은 태블로 알고리즘 기반의 추론 엔진이 실행되는 동안 지식 베이스(Knowledge Base)에 존재하는 공리들 간의 종속 관계를 파악함으로써 추론에 대한 과정을 보여준다. 태블로 알고리즘 기반 결정 모듈의 주된 목적은 ALC 수준 서술 논리의 TBox에 존재하는 비논리적 개념과 연관된 개념들을 표현하는 것 이다.

논리적 오류 원인탐지와 관련된 연구 중 가장 최근에 발표된 기법으로는 서론에서 언급한 종속 부호 기반 MUPS(Minimal Unsatisfiability Preserving Sub-TBoxes)구성 기법[5]이 있다. Aditya Kalyanpur가 제안한 이 방법은 지식 베이스에 존재하는 개념들 중 논리적 충돌을 발생시키는 최소화된 비논리적 공리 집합을 식별한다. 이 방법 역시 태블로 알고리즘 기반 결정 모듈을 바탕으로 하고 있으며 SHOIN 수준의 서술 논리를 처리할 수 있다.

본 장에서 언급한 기법들은 논리적 충돌을 유발하는 개념과 관련된 모든 공리들을 도출하여 표현한 뿐 논리적 오류의 근본적인 원인을 제공하는 공리는 식별하지 않는다. 따라서 본 논문에서 제안하는 기법은 이러한 제약성을 해결하는 것이 목적이다. non-deterministic 문

제 해결을 위한 최적화 기법의 관련 연구인 종속 부호 기반 백트랙킹은 3장에서 자세히 설명한다.

3. 기본 개념

3.1 온톨로지 추론과 태블로 알고리즘

웹 온톨로지 언어인 OWL[8]은 온톨로지를 구축하는데 사용된다. 온톨로지 모델링 관점에서 볼 때 OWL은 서술 논리[4]가 가지고 있는 많은 논리적 구성과 강하게 일치하고 있다. 서술 논리는 사람이 가지고 있는 지식을 기본적으로 개념(Concept), 관계(Role), 개체(Individual)로 표현하고, 이것을 TBox와 ABox로 나눈다. TBox에는 개념의 계층 관계와 정의(Terminological)부분이 선언되는데, 이것은 온톨로지의 클래스(Class)와 제약(Restriction) 정의 부분이다. ABox에는 개념의 개체(Assertional)부분이 정의되는데, 이것은 온톨로지의 개체(Instance) 정의 부분이다.

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. OWL을 위한 추론 서비스는 일반적으로 서술 논리에 대한 추론 서비스와 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 검사로 구성된다.

현재까지 온톨로지 추론 엔진을 구축하는 연구는 서술 논리로 표현된 온톨로지의 정확하면서 완전한 추론을 합리적인 시간 안에 수행할 수 있는 알고리즘 개발에 집중되고 있다. 태블로 알고리즘은 온톨로지 추론을 합리적인 시간 안에 수행할 수 있음을 실험적으로 증명하였고, 현재 많은 온톨로지 추론 기술은 태블로 알고리즘을 기반으로 하고 있다. 태블로 알고리즘은 초기 조건에서 해당되는 규칙을 실행시켜 레이블과 노드를 확장해 나가는 방식으로써, 확장되어지고 있는 레이블에 논리적 충돌(Clash)이 발생하면, 논리적으로 정당하지 않다(Unsatisfiable)라고 판정하고, 더 이상 규칙을 적용할 수 없을 때까지 논리적 충돌이 발생하지 않으면 논리적으로 정당(Satisfiable)하다고 판정한다.

태블로 알고리즘의 기본 아이디어는 증명하고자 하는 내용의 논리적 부정(Negation)에 대해서 다양한 변환 규칙을 적용하여 정당하지 않다는 것을 보여주는 방식을 취하고 있다. 즉, 서술 논리는 $\sqcap$, $\sqcup$, $\neg$, $\exists$, $\forall$등으로 표현되기 때문에 태블로 알고리즘에서는 이에 대한 확장 규칙을 반복적으로 적용하면서 탐색공간을 확장하게 된다. 그리고 모든 탐색 공간의 말단 노드가 모순(Contradiction)이라는 것을 보여줌으로써 증명하고자 하는 포함 관계의 논리적 부정이 정당하지 못하다는 것을 증명하게 된다. 노드가 모순이 되기 위해서는 같은 노드 레이블에 개념 C와 부정인 $\neg$C가 동시에 존재해야 한다. 표 1은 태블로 알고리즘에서 활용하는 기본 확장 규칙의 일부를 보여준다.

표 1 태블로 알고리즘

$\sqcap$-rule	if 1. $(C_1 \sqcap C_2) \in \mathcal{L}(x)$ 2. $\{C_1, C_2\} \not\subseteq \mathcal{L}(x)$ then $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C_1, C_2\}$
$\sqcup$-rule	if 1. $(C_1 \sqcup C_2) \in \mathcal{L}(x)$ 2. $\{C_1, C_2\} \cap \mathcal{L}(x) = 0$ then a. save T b. try $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C_1\}$ If that leads to a clash then restore T and c. try $\mathcal{L}(x) \rightarrow \mathcal{L}(x) \cup \{C_2\}$
$\exists$-rule	if 1. $\exists R.C \in \mathcal{L}(x)$ 2. there is no y s.t $\mathcal{L}(\langle x,y\rangle)=R$ and $C \in \mathcal{L}(y)$ then create a new node y and edge $\langle x,y\rangle$ with $\mathcal{L}(y)=\{C\}$ and $\mathcal{L}(\langle x,y\rangle)=R$
$\forall$-rule	if 1. $\forall R.C \in \mathcal{L}(x)$ 2. there is some y s.t. $\mathcal{L}(\langle x,y\rangle)=R$ and $C \notin \mathcal{L}(y)$ then $\mathcal{L}(y) \rightarrow \mathcal{L}(y) \cup \{C\}$
$\forall^+$-rule	if 1. $\forall R.C \in \mathcal{L}(x)$ 2. there is some y s.t. $\mathcal{L}(\langle x,y\rangle)=R$, Trans(R) 3. $R \sqsubseteq S$ 4. $\forall S.C \notin \mathcal{L}(y)$ then $\mathcal{L}(y) \rightarrow \mathcal{L}(y) \cup \{\forall S.C\}$
$\geq$-rule	if 1. $(\geq n\ R\ C) \in \mathcal{L}(x)$ 2. there are no $y_1 \ldots y_n$ of $y_i \neq y_j$ s.t $\mathcal{L}(\langle x,y_1\rangle)=R \ldots \mathcal{L}(\langle x,y_n\rangle)=R$ and $C \in \mathcal{L}(y_1) \ldots C \in \mathcal{L}(y_n)$ then 1. create new nodes $y_1 \ldots y_n$ and edge $\langle x,y_1\rangle \ldots \langle x,y_n\rangle$ with $\mathcal{L}(y_1)=\{C\} \ldots \mathcal{L}(y_n)=\{C\}$ 2. add $y_i \neq y_j$
$\leq$-rule	if 1. $(\leq n\ R\ C) \in \mathcal{L}(x)$ 2. there are $y_1 \ldots y_m$ s.t $\mathcal{L}(\langle x,y_1\rangle)=R \ldots \mathcal{L}(\langle x,y_n\rangle)=R$ and $C \in \mathcal{L}(y_1) \ldots C \in \mathcal{L}(y_n)$ 3. $1 \leq i,j \leq m$ and $m > n$ Then merge(y_i, y_j) in tableaux model

3.2 종속 부호 기반 백트랙킹

태블로 알고리즘의 논리합 규칙($\sqcup$-규칙)은 레이블에 $C_1 \sqcup C_2$가 존재할 경우 C_1을 레이블에 일단 추가하여 모순이 발생하면, 이전 상태로 환원하여(백트랙킹) C_2를 레이블에 추가하여 모순이 발생하는지 확인하는 non-deterministic 규칙이다. 따라서 레이블에 논리합으로 연결된 공리들이 많을수록, 논리적인 모순을 유발하는 공리와 상관없이 불필요한 백트랙킹이 발생한다.

예를 들어 노드 x의 레이블에 $\{(C_1 \sqcup D_1), \ldots, (C_n \sqcup D_n), \exists R.(A \sqcap B), \forall R.\neg A\}$와 같은 공리들이 있다고 했을 때, 일반적으로 논리합 규칙은 존재 한정 규칙($\exists$-규칙)나

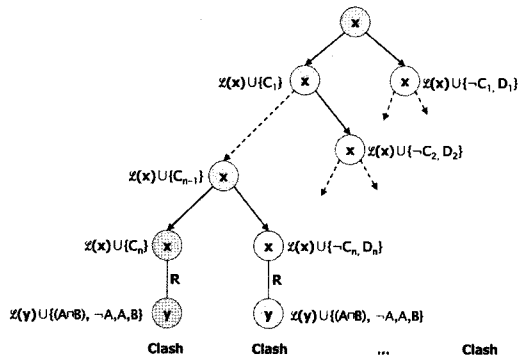

그림 1 태블로 알고리즘으로 인해 발생하는 non-deterministic 문제

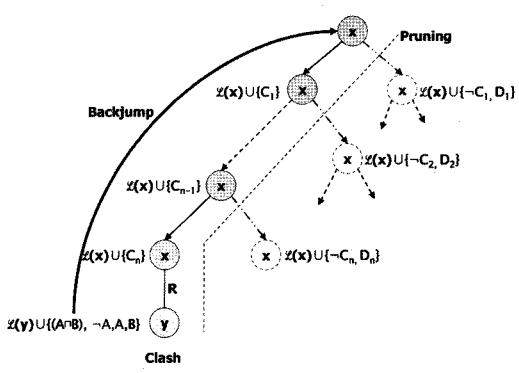

그림 2 태블로 알고리즘의 종속 부호 기반 백트랙킹을 적용에 따른 효율성

전체 한정 규칙(∀-규칙) 보다 우선순위가 높기 때문에 (탐색공간의 효율성) C_1에서 C_n까지 먼저 레이블에 확장하게 된다. 다음에 존재 한정 규칙, 전체 한정 규칙, 논리곱 규칙(⊓-규칙)을 차례로 적용하면 A와 ¬A에 의해 모순이 발생한다. 그림 1에서 보듯이 $(C_1 \sqcup D_1)$부터 $(C_n \sqcup D_n)$까지 논리합 규칙을 적용받는 공리는 모순에 영향을 미치지 않는다. 그럼에도 불구하고 논리적인 모순이 발생하게 되면 가장 최근의 논리합 규칙을 적용받은 분기점(C_n이 추가된 지점)으로 백트랙킹 하여 D_n 을 추가하여 계속 태블로 알고리즘을 진행하게 된다. 이렇듯 태블로 알고리즘은 non-deterministic 처리에 있어서 시간과 공간 복잡도가 높다.

불필요한 백트랙킹과 이에 따른 노드의 반복된 확장, 제거를 예방하기 위한 최적화 기법으로 종속 부호 기반 백트랙킹(Dependency Directed Backtracking-DDB)[4]이 발표되었다. 종속 부호 기반 백트랙킹은 non-deterministic한 상황이 발생할 때마다 분기점(Branch Point)을 공리의 종속 부호에 기록함으로써, 논리적 모순이 발생하였을 경우, 논리적 모순의 직접적인 원인을 제공하는 분기점으로 직접 백트랙킹 하는 기법이다. 그림 2는 앞선 예제에 종속 부호 기반 백트랙킹을 적용한 것을 보여준다. 초기 레이블에 존재하는 공리들인 $(C_1 \sqcup D_1),...,(C_n \sqcup D_n), \exists R.(A \sqcap B), \forall R.\neg A$은 분기점이 0이다. 여기에 논리합 규칙을 반복적으로 적용하면서 C_1에 1부터 C_n에 n이 분기점으로 할당된다. 그리고 존재 한정 규칙, 전체 한정 규칙, 논리곱 규칙을 차례로 적용하고, 모순을 일으키는 A와 ¬A의 분기점을 확인한다. 이때 두 공리의 분기점이 모두 0이므로 최상위 노드로 분기하며, 더 이상의 백트랙킹 없이 알고리즘을 중단한다.

다음 표 2는 종속 부호 기반 백트랙킹 알고리즘이다.

표 2 종속 부호 기반 백트랙킹 알고리즘

```
초기화: 최상위 노드에 존재하는 모든 공리의 종속부호의 분기점 집합에 0을 추가
        분기점 카운터를 1할당
if 태블로 알고리즘 중 deterministic 규칙이 적용될 때
  then 규칙이 적용된 후 레이블에 추가된 공리의 종속 부호에 규칙이 적용되기 전의 공리의 분 기점 집합을 복사
       if 논리적 모순이 발생하면
          then 모순을 발생시키는 공리들의 종속부호에 기록된 분기점 집합을 합집합(Union)으 로 구성한 후, 결과를
               반환(재귀호출 발생)
       if 논리적 모순이 발생하지 않으면
          then 0을 반환
if 태블로 알고리즘 중 non-deterministic 규칙이 적용될 때
  then 논리합으로 연결된 첫 번째 공리를 레이블에 추가
       레이블에 추가된 공리의 종속부호에 현재 분기점 카운터를 추가하고, 분기점 카운터 1증가
if 논리적 모순이 발생에 따른 재귀호출이 발생하면
  then 현재 분기점 카운터를 1 감소
       if 현재 분기점 카운터가 반환받은 분기점 집합에 속하지 않으면
          then 분기점 집합을 다시 결과로 반환(재귀호출 발생)
       if 현재 분기점 카운터가 반환받은 분기점 집합에 속하면
          then 분기점 집합에서 현재 분기점 카운트 값을 제거하고,
               두 번째 공리를 레이블에 추가
```

3.3 태블로 알고리즘 기반 결정 모듈

태블로 알고리즘 기반 결정 모듈을 이해하기 위해 MUPS (*Minimal Unsatisfiability Preserving Sub-TBoxes*) and 도출 과정(*Justification*)에 대한 개념을 이해해야 한다. 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 $K' \subseteq K$는 C의 *MUPS*가 된다. 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.

$$consequent^{[ante1,ante2,\ldots,anten]}$$

태블로 알고리즘 기반 결정 모듈은 SHOIN수준의 서술 논리에서 개념의 정당성을 증명하기 위해 태블로 알고리즘을 확장한 것이다. 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. 태블로 알고리즘 기반 결정 모듈의 목적은 입력된 온톨로지 내에서 논리적 충돌을 유발하는 공리들을 식별하는 것이다. 따라서 이 알고리즘은 다음 같이 MUPS를 구성하기 위해 표준 태블로 알고리즘 진행에 대해 3가지 변화를 주었다.

- 노드의 개념을 추가하는 것 외에 논리적 분기점 추가와 논리적 충돌에 원인이 되는 공리 삽입과 같은 명령들을 실행하기 위해 공리의 ID를 유지한다.
- 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 결합(Union)하고, 결합된 최종 결과물을 추가한다.
- 가능한 모든 논리적 충돌이 표시될 때까지 그래프를 탐색한다.

태블로 알고리즘 기반 결정 모듈은 기호 $\mathcal{G}$으로 알고리즘 진행 중 발생하는 모든 논리적 충돌 상황을 저장한다. 그래프 G은 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 상황에서 $(x \neq y)$가 레이블에 속할 때이다.

이 알고리즘은 그래프의 변화를 기록하기 위해서, 논리적 충돌 이벤트가 일어났을 경우 발생하는 논리적 분기점과 공리들을 종속적인 순서대로 도출 과정에 기록하는 과정 탐지 함수(*Tracing Function*)을 적용한다. 과정 탐지 함수 τ는 각 논리적 충돌 이벤트의 해당 집합으로 매핑 되는데, 각 집합은 논리적 분기 지점과 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.

4. 논리적 부정 가정 기반 진리 유지

가정 기반 진리 유지 시스템(ATMS)은 추론 엔진의 추론 과정을 기억하고 각 추론 상태의 진위를 관리하는 기능을 수행한다. 이는 TMS 네트워크를 구성하는 각 노드의 레이블과 노굿(Nogood)을 관리함으로써 추론 과정에서 논리적 모순이 발생하였을 때 이를 효과적으로 처리하는 것이다[9,10]. 여기서 노굿은 논리적 모순을 유발하는 가정들의 집합이다. 즉, 추론 엔진은 규칙이나 사례의 변수를 상수로 단일화(Unify)하여 도출 과정(Justification) 형태로 바꾸어 ATMS에 전달하고, ATMS는 이와 같은 도출 과정을 종속(Dependency) 구조로 표현하여 관리한다. 또한 ATMS는 추론 엔진에서 논리적 모순을 발생시키는 추론이 발생하는 경우에 이를 탐지하고 논리적 모순을 해결하는 기능을 가지고 있다. 예를 들어 "a,b $\rightarrow \perp$"라는 논리적 모순이 발생하는 규칙이 존재할 때, a와 b가 동시에 존재해야 모순이 발생하게 된다. 이때 ATMS는 a나 b중 하나 이상을 제거함으로써 모순을 해결한다. 이와 같은 방식은 대부분의 경우에 문제를 해결하는 것으로 알려지고 있다[10]. 그림 3

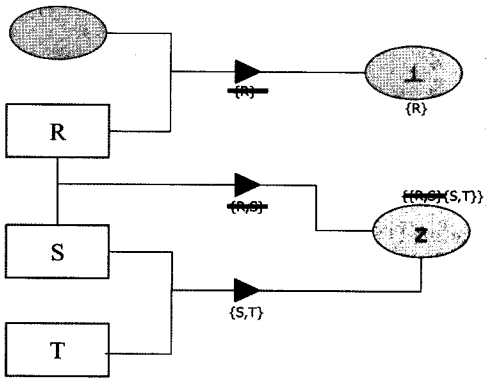

그림 3 ATMS 네트워크의 노굿 처리

의 ATMS 네트워크는 {R,S}가 Z를 추론하지만 {R}이 노굿이기 때문에 {R,S} 역시 노굿이 되고, 따라서 {R}과 {R,S}는 삭제된다. 즉, 추론 엔진에 의해서 추론이 실행된 경우에 ATMS는 모순을 탐지하고, 모순에 영향을 주는 가정들을 추론 엔진에 제시하여 제거할 가정을 알아내며, 그 가정이 영향을 준 노드들의 레이블을 관리한다[10]. 이와 같은 방식은 대부분 성공적 이지만 다음과 같은 사항이 네트워크에 존재하는 경우 정확한 노굿을 찾아내기가 불가능하다.

choose{A, B}, A,C ⇒ ⊥, B,C ⇒ ⊥

기본적인 ATMS는 혼 절(Horn clause)만을 허용하지만, 확장 ATMS (Full ATMS)는 가정들이 논리합으로 연결된 긍정절(Positive clause)까지 허용하며, 입력값으로 혼 절로 표현된 도출 과정과 긍정절로 표현된 choose를 받을 수 있다. 따라서 choose{A, B}는 A∨B로 표현할 수 있다. 위의 경우 A와 B는 믿음값(belief value)으로 주어졌으므로 실제로 노굿은 {C}가 된다. 그러나 [9,10]에서 제시하고 있는 ATMS 알고리즘은 {C}가 노굿이라는 것을 탐지하지 못한다.

이러한 사항을 해결하기 위해서 Full ATMS는 choose{A, B}를 A∨B, A,C⇒⊥를 ¬A∨¬C, B,C⇒⊥를 ¬B∨¬C로 부호화(Encoding)하여, 분해(Resolution)통해 ¬C를 유도한다. 즉 ¬C는 C⇒⊥으로 복호되기 때문에 {C}가 노굿인 것을 탐지한다. 그러나 분해를 사용하면, 시간 복잡도가 높아지고, 모순과 관계없는 가정들과 추가적인 도출 과정의 부호화를 요구하기 때문에, 다른 응용 시스템에 적용하기가 적절치 않다.

논리적 부정 가정 기반 진리 유지 시스템(Negated Assumption based Truth Maintenance System - NATMS)[11]은 분해를 사용하지 않고 {C}가 노굿이라는 것을 탐지한다. NATMS는 추론 엔진으로부터 전달받은 도출 과정의 구조 중 전제 조건을 구성하는 리터럴에 논리적 부정을 허용한다. 따라서 추론 엔진은

NOGOOD(E)

1. Mark *E* as nogood.
2. Remove *E* and any superset from every node label
3. [Handle negated assumptions] For every A ∈ E for which ¬A appears in some justification call **UPDATE({E - {A}}, ¬A)**.

그림 4 NATMS의 노굿 관리 알고리즘

choose{A, B}를 ¬A∧¬B⇒⊥의 도출 과정 형식으로 바로 전달할 수 있다. 그림 4는 논리적 부정이 허용된 가정을 사용하여 노굿을 관리하는 알고리즘이다. 위의 사항을 알고리즘에 적용하는 예를 들었을 때, NATMS 네트워크에는 두 개의 노굿 {A,C}, {B,C}이 존재한다. 이때 ¬A∧¬B⇒⊥를 추론 엔진으로부터 전달 받게 되면 ¬A와 ¬B의 논리적 부정인 A와 B가 노굿 {A,C}, {B,C}내에 존재하므로 ¬A와 ¬B의 가정은 C가 되며, ¬A∧¬B⇒⊥로부터 ⊥의 레이블에 {C}가 추가된다. 따라서 ⊥의 레이블에는 {A,B},{B,C},{C}가 존재하며, 이 중 {A,B},{B,C}는 {C}와 포함관계(Subsume)를 갖기 때문에 생략된다. 최종적으로 {C}가 정확한 노굿이 된다.

5. 논리적 부정 가정 기반 진리 유지 시스템을 적용한 백트랙킹과 논리적 오류 탐지 기법

본 논문에서는 논리적 부정 가정 기반 진리 유지 시스템(Negated Assumption based Truth Maintenance System -ATMS)을 사용하여 non-deterministic 문제를 해결하고 논리적 오류 원인을 탐지하는 기법을 제안한다. 기존에 발표되었던 기법들의 공통점은 종속 부호(Dependency Label)를 사용한다는 것이다. 종속 부호는 태블로 규칙이 적용될 때마다 레이블에 추가되는 공리를 도출하는 공리들이 종속적인 순서대로 기록된 자료구조이다. NATMS 역시 추론 엔진에 의해서 추론된 명제를 전달받아 상호 의존 관계를 그래프로 표현한다. 따라서 태블로 알고리즘의 추론 분기점을 탐지하거나 논리적 모순을 일으키는 원인을 찾아내는데 유용하게 사용될 수 있다.

본 논문에서 제안하는 방법은 다음과 같은 장점을 갖는다. NATMS는 상호 의존 관계를 그래프로 표현하므로 non-deterministic 문제를 해결하는 동시에 논리적 오류에 원인이 되는 공리들을 탐지한다. 이는 온톨로지 디자이너 또는 온톨로지 구축 프로그램이 온톨로지를 디버깅할 때 좋은 기본 자료가 된다.

5.1 NATMS 프로토콜

온톨로지 추론 엔진에 NATMS를 적용하기 위해서 먼저 두 시스템간의 프로토콜을 정의하였다. 프로토콜은 태블로 알고리즘의 규칙들의 특징을 기반으로 만들어졌

다. 표 3은 정의된 프로토콜을 설명한다.

- ⊓- 규칙 - 추론 엔진은 $C_1 \sqcap C_2$ 가 존재할 경우 C_1, C_2를 레이블에 추가한다. 따라서 $C_1 \sqcap C_2 \rightarrow C_1$, $C_1 \sqcap C_2 \rightarrow C_2$로 도출 과정을 구성하여 NATMS에 전달한다.
- ⊔- 규칙 - 추론 엔진은 $C_1 \sqcup C_2$ 가 존재할 경우 C_1, C_2를 레이블에 번갈아 가면서 추가한다. 따라서 $A(C_1)$, $A(C_2)$를 번갈아 가면서 NATMS에 전달한다.
- ∃- 규칙 - 추론 엔진은 $\exists R.C_1$이 존재할 경우 새로운 R-이웃 노드를 생성하고, 생성된 노드 레이블에 C_1을 추가한다. 따라서 먼저 (Create NATMS R 1)을 NATMS에 전달하여 새로운 NATMS 네트워크를 선언하고 $A(C_1)$을 전달한다.
- ∀- 규칙 - 추론 엔진은 $\forall R.C_2$가 존재할 경우 R-이웃 노드 레이블에 C_2를 추가한다. 따라서 먼저 (Change NATMS R 1)를 NATMS에 전달하여, 변경된 NATMS로 이동을 한 후, $A(C_2)$를 전달한다.
- ∀+- 규칙 - 추론 엔진은 $R \sqsubseteq S$인 $\forall R.C_2$가 존재할 경우 R-이웃 노드 레이블에 $\forall S.C_2$를 추가한다. 따라서 먼저 (Change NATMS R #n)를 NATMS에 전달하여, 변경된 NATMS로 이동을 한 후, $A(S.C_2)$를 전달한다.
- ≥- 규칙 - 추론 엔진은 (≥ n R C)가 존재할 경우, 새로운 R-이웃 노드를 생성하고, 생성된 노드 레이블에 C를 추가한다. 따라서 먼저 (Create NATMS R n)을 NATMS에 전달하여 새로운 NATMS 네트워크를 선언하고 A(C)을 전달한다.
- ≤- 규칙 - 추론 엔진은 (≤ n R C)가 존재할 경우 추론 엔진에서 통합한 노드의 &key 값을 갖는 두 NATMS 네트워크를 통합한다. 따라서 (Merge NATMS &key1 &key2)를 NATMS에 전달한다.
- unfold- 규칙 - 추론 엔진은 C_1이 D_1으로 확장되는 정보가 논리 확장 맵(unfold-맵)에 존재할 경우 D_1을 레이블에 추가한다. 따라서 $C_1 \rightarrow D_1$을 NATMS에 전달한다. NATMS는 도출 과정의 결론부분은 논리적 부정을 허용하지 않기 때문에, 만약 C_1이 $\neg D_1$으로 확장된다면, $C_1 \sqcap D_1 \rightarrow \perp$를 NATMS에 전달한다.
- 모순 발생 - 추론 엔진은 C_1이 레이블에 존재하는 상황에서 $\neg C_1$이 추가가 되면 모순을 발생시킨다. 따라서 choose(C_1)을 구성하여 NATMS에 전달한다. 이때 choose(C_1)은 최종적으로 C_1이 모순을 발생시키기 때문에 C_1이 믿음 값임을 의미한다.

5.2 NATMS 기반 논리적 오류 원인과 분기점 탐지 과정

태블로 알고리즘에서의 논리적 충돌은 C와 ¬C가 동시에 존재하는 것을 의미한다. C가 믿음값일 경우, ¬C는 거짓이 된다. 따라서 추론 엔진에서 논리적 충돌을 일으키는 공리C가 주어지면, ¬C를 도출하는 개념이 논리적 오류 원인과 분기점이 된다.

논리적 오류 원인과 분기점을 탐지하는 과정은 다음과 같다. 먼저 A와 B가 논리적 충돌을 일으키는 공리로 주어지고,

$$\text{choose}(A,B) \Rightarrow \neg A \wedge \neg B \rightarrow \perp$$

⊥의 가정 집합(노굿)이 {{A,C},{B,C}}이라면, 그림 4에 표현된 NATMS 노굿 관리 알고리즘에 의해서 다음과 같이 표현할 수 있다.

$$A \wedge C \rightarrow \perp,\ B \wedge C \rightarrow \perp \Rightarrow \langle \neg A, \{\{C\}\}\rangle,\ \langle \neg B, \{\{C\}\}\rangle$$

$\neg A \wedge \neg B \rightarrow \perp$이기 때문에, ⊥의 가정 집합(노굿)이 {{A,C},{B,C},{C}}이며, 최종적으로 {C}는 {A,C},{B,C}에 포함되므로 ⊥의 노굿은 C가 되며, 논리적 오류의 원인과 분기점이 된다.

5.3 NATMS를 기반으로 논리적 오류 원인과 이와 연관된 분기점을 탐지하는 예제

태블로 규칙을 적용하면서 레이블에 새로운 개념이 추가되거나 논리적 충돌을 발생시키는 공리가 탐지되면, 이에 해당하는 프로토콜을 생성하여 NATMS에 전달한다. NATMS는 전달받은 프로토콜을 바탕으로 NATMS 네트워크를 구축하면서 노굿을 식별한다. 그림 5는 non-deterministic 문제를 발생시키는 공리에 대해서 NATMS를 기반으로 논리적 오류와 연관된 분기점을 탐지하는 예를 보여준다. 온톨로지 추론 엔진이 개념 *Evan*의 논리적 정당성을 검사한다고 가정한다. 먼저 개념 *Evan*의 논리 확장 정보가 $C \sqcup D$이므로 개념 C가

표 3 온톨로지 추론엔진과 NATMS간의 프로토콜

프로토콜	설명
(create-natms &key #n)	새로운 NATMS 네트워크를 생성
(change-natms &key #n)	명시된 Key값을 갖는 NATMS 네트워크로 이동
(Merge-natms &key1 &key2)	두 Key값을 갖는 NATMS 네트워크를 합병
A(axiom)	⊓,⊔,∃,∀,∀+, ≥ 규칙 적용에 따라 레이블에 추가되는 공리 NATMS 네트워크에 새로운 노드를 생성
Justification(unfold_process)	⊓와 unfold 규칙 적용에 의해 공리가 확장되는 과정을 도출과정으로 정형화
choose(axiom)	모순을 발생시키는 공리를 choose로 정형화
WhatNogood?	현재 정확한 노굿을 질의

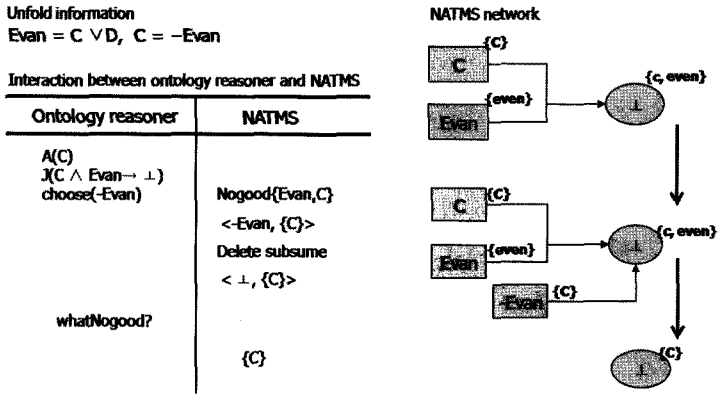

그림 5 NATMS를 기반으로 논리적 오류와 연관된 분기점을 탐지하는 예

논리합 규칙에 의해 레이블에 추가되면서 프로토콜 A(*C*)가 전달된다. NATMS는 가정이 {C}인 새로운 노드 C를 생성하여 네트워크에 추가한다. 개념 C의 논리 확장 정보가 ¬*Even*이므로 논리 확장 규칙이 적용되면서 도출 과정 $C \sqcap Evan \rightarrow \bot$이 생성되어, 프로토콜 J($C \sqcap Evan \rightarrow \bot$)가 전달된다. ⊥의 가정이 {*C*, *Evan*}이므로, {*C*, *Evan*}이 노굿이 된다. *Evan*과 ¬*Evan*이 논리적 충돌을 발생시키므로, 프로토콜 choose(*Evan*)이 전달되고, ¬*Evan*이 ⊥의 종속 부호에 존재하기 때문에 노굿 관리 알고리즘에 의해 ¬*Evan*의 가정은 {*C*}가 된다. choose(*Evan*)의 의미는 ¬*Evan*→⊥이므로, ⊥의 종속 부호에 {*C*, *Evan*}, {*C*}가 기록되는데, 이때 {*C*, *Evan*}는 {*C*}와 포함 관계를 가지므로 ⊥의 종속 부호에서 제거된다. 최종적으로 ⊥의 가정이 {*C*}가 되기 때문에 개념 *C*가 논리적 오류와 연관된 분기점이 되는 동시에 논리적 오류를 유발하는 원인이 된다.

그림 6은 NATMS를 기반으로 non-deterministic 문제가 없는 개념의 논리적 오류 원인을 탐지하는 예를 보여준다. 그림 6은 온톨로지 추론 엔진이 개념 *Evan*의 논리적 정당성을 검사한다고 가정한다. 먼저 개념 *Evan*의 논리 확장 정보가 $C \sqcap A$이므로, 개념 C와 A가 논리곱 규칙에 의해 레이블에 추가되면서, 프로토콜 A(*C*)와 A(*A*)가 NATMS에 전달되고, NATMS는 가정이 {C}인 노드 C와 가정이 {A}인 노드 A를 생성하여 네트워크에 추가한다. 개념 *A*의 논리 확장 정보가 *B*이므로 논리 확장 규칙이 적용되면서 도출 과정 *A*→*B*가 생성되어, 프로토콜 J(A→B)가 전달된다. 따라서 노드 *B*의 가정은 {*A*}가 된다. 다음 개념 *B*의 논리 확장 정보가 ¬*C*이므로 논리 확장 규칙이 적용되면서 도출 과정 $B \sqcap C \rightarrow \bot$이 생성되며, 프로토콜 Justification($B \sqcap C \rightarrow \bot$) 가 전달된다. 따라서 ⊥의 도출 가정이 *B*와 *C*이므로, {*A*, *C*}가 노굿이 된다.

*C*와 ¬*C*가 논리적 충돌을 발생시키므로, 프로토콜 choose(C)가 전달되고, ¬*C*가 ⊥의 종속 부호에 존재하기 때문에 노굿 관리 알고리즘에 의해 ¬*C*의 가정은 {A}가 된다. choose(*C*)의 의미는 ¬*C*→⊥이므로, ⊥의 종속 부호에 {*C*, *A*}, {*A*}가 기록되는데, 이때 {*C*, *A*}는 {*A*}와 포함 관계를 가지므로 ⊥의 종속 부호에서 제거

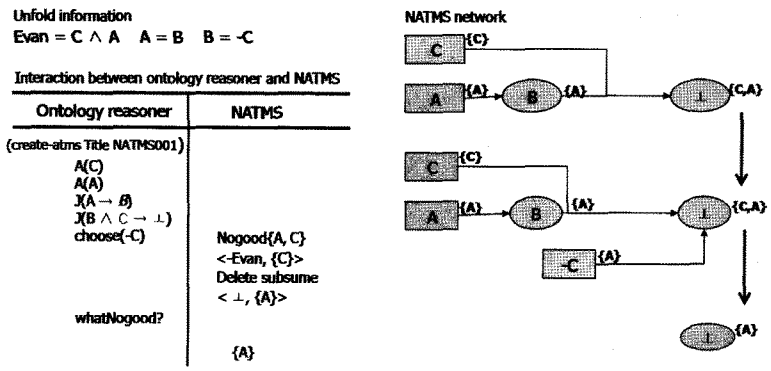

그림 6 NATMS를 기반으로 논리적 오류의 원인을 탐지하는 예

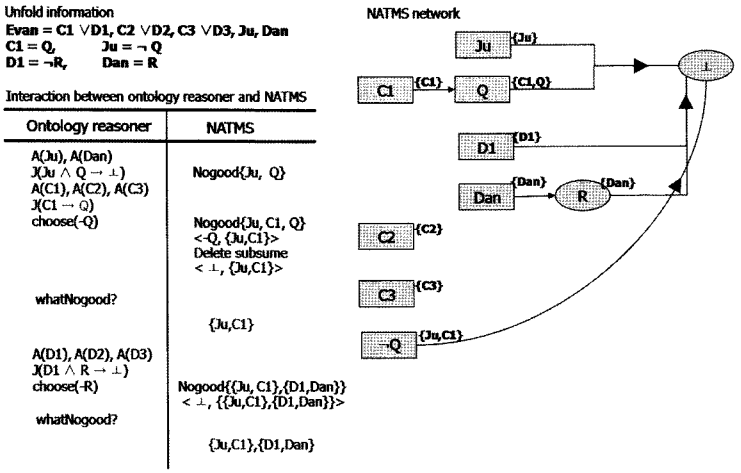

그림 7 NATMS를 기반으로 논리적 오류와 연관된 분기점을 탐지하는 예

된다. 최종적으로 ⊥의 가정이 {A}가 되기 때문에 개념 C가 논리적 오류와 연관된 분기점이 되는 동시에 논리적 오류를 유발하는 원인이 된다. 그림 7은 non-deterministic 문제를 발생시키는 공리에 대해서 NATMS를 기반으로 논리적 오류와 연관된 분기점을 탐지하는 복잡한 예를 보여준다.

6. 실험 및 평가

본 장에서는, 4장과 5장에서 설명한 논리적 부정 가정 기반 진리 유지 시스템(NATMS)을 적용하여 온톨로지의 non-deterministic 문제 해결과 논리적 오류를 유발하는 개념 탐지 기법에 대해서 실험 평가 한다.

본 논문에서 제안한 기법은 대표적인 서술 논리 추론 엔진인 Pellet에 적용되었다. 표 4는 실험에 사용된 온톨로지의 상세 정보를 보여주고 있다. 실험에 사용한 온톨로지는 Swoop 프로젝트[12]에서 사용한 온톨로지들로 선택하였으며 펜티엄 CPU(1.77GHZ)와 메모리(1G)를 장착한 PC에서 실행하였다.

Pellet은 SHOIQ 수준의 표현력을 지닌 온톨로지의 T-Box와 A-Box 추론 기능을 지원한다. 따라서 본 논문에서 제안한 온톨로지의 non-deterministic 문제 해결과 논리적 오류를 유발하는 개념 탐지 기법에 대한 성능 평가를 위해 제안한 기법이 적용된 Pellet과 DDB가 적용된 Pellet의 논리적 정당성 검사를 위한 실행 시간을 비교하였다.

표 5는 두 기법이 각각 적용된 Pellet이 온톨로지의 논리적 정당성을 검사하는데 소요한 시간을 보여준다. 실험 결과를 정리하면, 전체적으로 DDB 기법을 적용한 Pellet 보다 제안한 기법을 적용한 Pellet의 실행 속도가 비슷하거나 약간 빠른걸 알 수 있다. newChemical 온톨로지와 DOLCE 온톨로지의 경우 DDB 기법을 적용한 Pellet이 더 좋은 실행 속도를 보였는데, 두 온톨로지의 경우 많은 존재 한정자와 최소 한정으로 연결된 공리들이 다른 온톨로지에 비해 상당히 많았다. 본 논문에서 제안한 기법은 존재 한정 규칙(∃-rule)과 최소 한정(≤-rule)이 실행될 때 마다 새로운 NATMS 네트워크를 생성하기 때문에, 온톨로지 내에 엄청나게 많은 존

표 4 실험에 사용된 온톨로지 정보

Ontology	Axioms	Classes/ Unsatisfiable classes
Person	112	70/1
Wine	856	77/-
University	169	30/8
Chemical	254	48/37
DOLCE	1417	200/-
Economy	1704	338/51
Galen	6580	2749/-
Sweet-JPL	3833	1537/1
Tambis	800	395/144
Transport	2051	444/55

표 5 논리적 정당성 검사에 소용되는 시간(ms)

Ontology	Pellet(DDB)	Pellet(NATMS)
Person	412ms	413ms
Wine	2012ms	1802ms
University	314ms	307ms
newChemical	1021ms	1023ms
DOLCE	3415ms	3621ms
Economy	3567ms	2145ms
Galen	21752ms	15462ms
Sweet-JPL	1526ms	1421ms
Tambis	891ms	896ms
Transport	14653ms	14221ms

재 한정자와 최소 한정으로 연결된 공리들이 존재할 경우 다소 속도가 떨어졌다. 이 부분에 대해서는 향후 계속 연구 될 것이다.

7. 결 론

온톨로지 추론 엔진의 대부분은 태블로 알고리즘을 기반으로 구축되었다. 그러나 태블로 알고리즘은 non-deterministic 처리에 있어서 시간과 공간 복잡도가 높기 때문에, 이를 해결해 줄 최적화 기법이 필요하다. 또한 대부분의 태블로 알고리즘 기반의 추론 엔진들은 논리적 오류를 일으키는 원인은 밝히지 않고, 논리적 오류를 갖는 개념만을 탐지한다. 따라서 본 논문에서는 논리적 부정 가정 기반 진리 유지 시스템(NATMS)을 사용하여 non-deterministic 문제를 해결하고 논리적 오류 원인을 탐지하는 기법을 제안하였다.

가정 기반 진리 유지 시스템(ATMS)은 추론 엔진의 일부분으로서 추론 엔진의 추론 과정을 기억하고 각 추론 상태의 진위를 관리하는 기능을 수행한다. 또한 ATMS는 추론 엔진에서 논리적 모순을 발생시키는 추론이 발생하는 경우에 이를 탐지하고 논리적 모순을 해결하는 기능을 가지고 있다. 그러나 기본적인 ATMS는 혼 절(Horn clause)만을 허용하기 때문에 정확한 노굿을 판별하기 힘들다. NATMS는 추론 엔진으로부터 전달받은 도출 과정 중 전제조건을 구성하는 리터럴에 논리적 부정을 허용한다. 따라서 노굿을 판별하기 위해 ATMS가 사용하는 분해를 적용하지 않고서 바로 정확한 노굿 판별이 가능하다.

본 논문에서 온톨로지 추론 엔진에 NATMS를 적용하기 위해 먼저 두 시스템간의 프로토콜을 정의하였다. 온톨로지 추론 엔진은 태블로 규칙을 적용하면서 레이블에 새로운 개념이 추가되거나 논리적 충돌을 발생시키는 공리가 탐지되면, 이에 해당하는 프로토콜을 생성하여 NATMS에 전달한다. NATMS는 전달받은 프로토콜을 바탕으로 NATMS 네트워크를 구축하면서 노굿 즉 논리적 오류와 연관된 분기점을 식별한다.

본 논문에서 제안한 기법은 존재 한정 규칙이 실행될 때 마다 새로운 NATMS 네트워크를 생성하기 때문에, 온톨로지 내에 엄청나게 많은 존재 한정자로 연결된 공리들이 존재할 경우 다소 속도가 떨어졌다. 이 부분에 대해서는 향후 지속적으로 연구될 것이다.

참 고 문 헌

[1] Dmitry Tsarkov and Ian Horrocks, FaCT++ description logic reasoner: System description, In Proc. of the Int. Joint Conf. on Automated Reasoning, IJCAR, 2006.

[2] Evren Sirin, Bijan Parsia, Bernardo Cuenca Grau, Aditya Kalyanpur and Yarden Katz, Pellet: A practical OWL-DL reasoner, Journal of Web Semantics, 2007.

[3] U. Hustadt, B. Motik U. Sattler, Reasoning in Description Logics with a Concrete Domain in the Framework of Resolution, Proc. of the 16th European Conference on Artificial Intelligence (ECAI 2004), Valencia, Spain, pp. 353-357, August, 2004.

[4] F. Baader and W. Nutt, *The Description Logic Handbook: Theory, Implementation, and Applications*, pp. 43-95. Cambridge University Press, 2003.

[5] Aditya Kalyanpur, Bijan Parsia, Bernardo Cuenca Grau and Evren Sirin, Justifications for Entailments in Expressive Description Logics, Technical report, 2006.

[6] McGuinness, D. and Borgida, A., Explaining Subsumption in Description Logics, Proceedings of the Fourteenth International Joint Conference on Artificial Intelligence, pp. 816-821, 1995.

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

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

[9] Johan de Kleer, Problem solving with the ATMS, Artificial Intelligence 28, pp. 163-196, 1986.

[10] Kenneth D. Forbus, Johan de Kleer, Building Problem Solvers, The MIT Press, 1993.

[11] Johan de Kleer, A General Labeling Algorithm for Assumption based Truth Maintenance, Proceedings of the AAAI-88, 1988.

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

김 재 민
정보과학회논문지 : 소프트웨어 및 응용
제 36 권 제 2 호 참조

박 영 택
정보과학회논문지 : 소프트웨어 및 응용
제 36 권 제 2 호 참조