퍼지 및 지능시스템 학회 2003, Vol. 13, No. 5, pp. 505-511

개선된 쌍 검증 방식을 이용한 지식 검증 시스템

Knowledge Verification System with Improved Pairwise Checking Method

서 의 현
Euyhyun Suh

목원대학교 컴퓨터공학과

요 약

생성 규칙의 지식 표현 방식은 많은 장점에도 불구하고 일관성 유지가 어렵다는 단점을 갖고 있다. 그런데 지식의 일관성 유지 여부는 지식을 기반으로 한 추론 결과의 신뢰도에 직결되므로 신뢰성을 제고하기 위하여 일관성 검증 시스템이 필수적이다. 일관성 검증 방식 중 가장 많이 사용되는 쌍 검증(pairwise checking) 방식에서는 가치 있는 규칙이 생략될 수 있으며 규칙의 수가 많은 경우에는 검증 시간이 많이 걸리는 단점이 있다. 따라서 본 논문에서는 확실한 특성의 리스트와 가능한 특성의 리스트를 사용하고 검증단계를 개선함으로써 쌍 검증 방식의 단점을 보완하여, 구조적 및 의미적 오류도 제거할 수 있는 검증 시스템을 제안하고 구축한다.

Abstract

Production rule based knowledge representation method has many advantages, but has the difficulties in maintaining the consistency of knowledge. Since the consistency maintenance of knowledge exercises a marked effect on the reliability of inference results, the system for consistency maintenance of knowledge is indispensable to increase the reliability. In the most popular pairwise checking method among consistency verification methods, the valuable rules can be omitted and it takes much time in checking the consistency when the rules are numerous. So, this paper is to propose and implement the verification system which can remove the structural errors and semantic ones, making up for the defects of pairwise checking method by using the certain property list and eventual property list and improving the steps of verification.

Key Words : 지식베이스, 일관성, 쌍 검증 방식, 지식검증시스템

1. 서 론

지식의 표현 방식 중 생성 규칙은 가장 많이 쓰이는 방식으로써 지식을 모듈화하여 나타낼 수 있고 지식을 독립적으로 추가하거나 삭제하고 변경할 수 있는 장점을 갖고 있다. 또한 다양한 종류의 지식을 표현할 수 있고 저장된 지식을 비절차적인 방식으로도 사용할 수 있는 장점도 있다.

그런데 지식은 방대한 양이 저장 될 경우가 많기 때문에 지식 제공자(전문가)나 지식 공학자가 내용 전체를 파악하기가 어렵고, 전문가는 때때로 직관적으로 생각하므로 추론에서 많은 과정을 생략한 채 지식을 지식베이스에 삽입할 수 있다. 또한 지식베이스는 점진적인 방식으로 오랜 기간에 걸쳐 구축되고 여러 명의 전문가들이 지식을 함께 저장하기 때문에 지식의 중복, 순환 및 모순의 문제가 야기될 수 있다. 이러한 문제들을 해결하기 위해서는 지식 제공자가 지식베이스에 지식을 저장하기 전에 정확성을 검증하여 발견된 오류를 자체 수정하거나 자체 수정이 불가능할 경우 지식 제공자에게 문의하여 수정하는 일관성 유지 시스템이 요구된다.

접수일자 : 2003년 2월 19일
완료일자 : 2003년 9월 4일

그러나 생성 규칙으로 표현된 지식의 일관성 검증 문제는 NP-복잡성 문제이다. 시간 복잡도가 최악의 경우 지수승의 형태로 나타내어진다. 따라서 지식베이스의 일관성 검증을 효율화 하기위한 많은 시도가 있었다. 이 중 쌍 검증 방식은 사용하기 쉬워 널리 사용되고 있으나 경우에 따라서는 가치 있는 규칙이 생략되고 의미 없는 규칙이 취급되는 일이 발생할 수 있으며 규칙의 수가 많아질 경우 시간이 많이 걸리는 단점이 있다[7].

따라서 본 논문은 확실한 특성의 리스트와 가능한 특성의 리스트를 생성하고 검증 방법 및 단계를 개선하는 형태로 쌍 검증 방식을 보완하여 지식이 지식베이스에 첨가될 때 지식의 정확성 및 일관성을 보장하기 위한 지식 검증 시스템을 구축한다. 지식 컴파일러에 의해 부정의 프레미스를 포함한 네트워크로 지식을 표현하고 보완된 쌍 검증 방식을 이용하여 구조적 측면에서뿐만 아니라 의미적 측면에서도 중복, 모순, 순환의 일관성 오류는 물론 도달될 수 없는 규칙과 더 이상 연결되지 않는 규칙 등의 완결성 오류를 찾아내어 자체 수정하거나 자체 수정이 불가능할 경우 전문가에게 수정하도록 메시지를 전달하는 지식 검증 시스템을 제시, 구축한다. 모든 시스템은 C++언어와 Oracle SQL을 사용하여 Sun Sparc 5.5에 구축되었다.

기존 시스템의 분석 및 문제점을 2장에서 설명한 후 3장

에서는 제안하는 시스템의 특징 및 동기에 대해 소개한다. 4장에서는 구현한 시스템 및 사례를 보여주고 5장에서는 시스템을 시간적 측면에서 성능 분석한다. 마지막으로 결론에서는 지식 검증 시스템을 구축함으로써 확인된 결과를 요약한다.

2. 기존 시스템 분석 및 문제점

NP-복잡성 문제인 생성 규칙으로 표현된 지식베이스의 일관성 검증을 위한 많은 시도가 있었다.

지식 기반 경영 시스템(Knowledge Based Management System :KBMS) 내에 일관성을 제약하는 조건을 제시하고, 제약 조건을 위반했을 경우 예외 처리를 실행하기 위하여 실행 형태 제어 규칙(Activation Pattern Controlled Rule)을 사용한 방식이 있다[5]. 그런데 이 방식은 부분적인 일관성만을 자동으로 처리할 수 있을 뿐이다.

의존 관계표를 사용하여 일관성과 완결성을 점검하는 방법[14]은 열거 방식으로 처리하기 때문에 시간이 많이 걸리는 단점이 있다.

결정표 (Decision Table)를 사용하는 방식[3]에서는 간혹 오류의 존재만을 발견하고 오류의 위치를 정확하게 찾아내지 못하는 단점이 있다. 또한 복잡한 관계를 가지는 규칙들은 추론 과정 중 연결된 규칙이 많아 지수 함수적인 성능을 나타낸다[11].

쌍 검증(Pairwise Checking) 방식에서는 생성된 절들이 적용 가능한 규칙들과 비교된다[7][10]. 이 때 경우에 따라서는 가치 있는 규칙이 생략되고 의미 없는 규칙이 취급되는 일이 발생할 수 있으며 규칙의 수가 많아질 경우 시간이 많이 걸리는 단점이 있다[8].

그래프 표현 방식은 프레미스의 개념적인 의존 관계를 나타내기 쉬운 표현법으로써 방향 그래프 (Directed Graph)[12], 추론 그래프 (Inference Graph)[15], 하이퍼 그래프(Hypergraph)[18]등의 방법이 있다. 그러나 이러한 방법들은 복합절과 단순절 사이의 의존 관계를 나타내기 어렵다[1][16]. 따라서 그래프 표현 방식은 노드를 첨가하거나 전이를 묘사하고 또는 하이퍼 아크를 사용하여 복합절을 묘사함으로써 중복과 포함을 정확히 발견하여 수정한다. 그렇지만 모순과 불완전성은 여전히 완벽하게 처리되지 못하는 단점이 있다. 방향 하이퍼 그래프 (Directed Hypergraph)[16]는 이러한 그래프의 단점을 보완하였지만 고려해야 할 하이퍼 노드의 조합의 수가 많아지면 실제로 이들을 다 묘사하지 못함으로써 오류가 생성될 수도 있다. 또한 이 방식은 구조적 오류만을 검색할 수 있을 뿐이며 규칙이 많아질 경우에는 시간이 많이 소요된다. 개념 그래프(Conceptual Graph)는 계층적 구조로써 지식을 체계적으로 분류하여 계층적으로 저장하고 편집 방법을 통해 중복을 제거한다[4][6]. 그러나 이 방식은 표현의 다양성 때문에 극히 제한적인 일관성만을 검진할 뿐 여러 곳에 있는 비 일관성 오류를 점검하기 어렵다.

페트리 네트(Petri Net)를 이용한 방식[8][13]은 일관성 검진이 정해진 실행 순서로 진행되어야 하는데 형식이 완전하지 못하면 제약 조건이 적용되지 못하고 수행시간도 많이 걸리는 단점이 있다.

지역 탐색 방식[9][17][2]은 매우 큰 지식 베이스에서 일관성 검증에 효율적인 성능을 보이지만 전체의 탐색 영역을 모두 고려하지 못하고 지식베이스가 비일관성 상태에 있다는 것을 직접 증명할 수 없다는 단점이 있다.[2]

따라서 본 논문은 전체 지식베이스의 영역을 고려하면서도 수행 시간을 단축하기 위하여 기본 구조 및 검증 방법을 개선시킨 쌍 검증 방식을 이용한 지식 검증 시스템을 제안한다.

3. 제안하는 시스템의 특징

제안하는 시스템은 (그림 1)과 같이 구성되며 크게 지식베이스, 지식 컴파일러와 지식 검증 시스템으로 나뉜다. 지식 검증 시스템에서 사용할 지식은 지식베이스 내에 생성규칙으로 표현되어져 있다. 이 지식은 지식 검증 시스템이 실행되는 동안 효율적이고 편리하게 사용될 수 있도록 지식컴파일러에 의해 네트워크 방식의 내부 표현방식으로 컴파일된다.

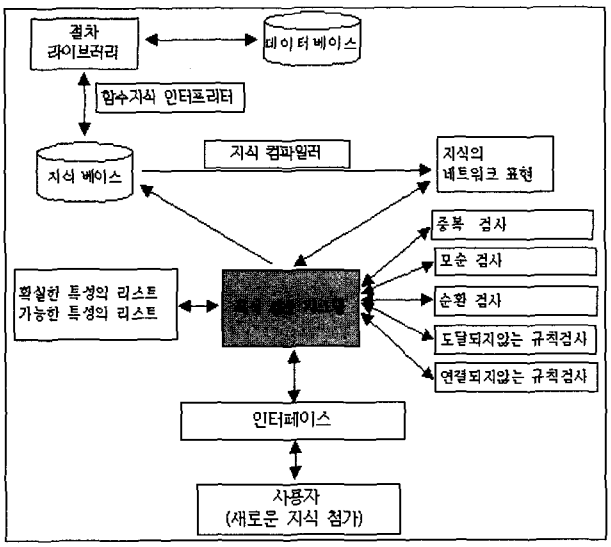

그림 1. 지식 검증 시스템
Fig.1. knowledge verification system

3.1 지식베이스

지식베이스의 기본 구조는 0+ 차원의 생성 규칙이다. 생성규칙은 규칙들을 모듈화하여 나타낼 수 있고, 비 절차적 방식으로도 규칙들로써 저장된 지식을 사용할 수 있을 뿐 아니라 여러 종류의 지식을 쉽게 표현할 수 있기 때문에 지식 베이스의 기본구조로써 생성 규칙을 선택했다. 지식은 조건 부분과 실행 부분으로 표현되며 일반 지식과 절차적 지식도 같은 방식으로 나타낼 수 있도록 하였다.

그리고 일반 사실적 정보는 정보의 방대함으로 인하여 관계형 데이터베이스 Oracle 7.3.3에 저장되어 있는 정보를 사용하도록 하였다. 필요한 경우에는 데이터베이스에 저장되어 있는 정보들을 호출하여 추론을 위한 지식으로 사용할 수 있다. 또한 데이터베이스 내에 삽입, 제거 및 수정할 수 있는 기능도 포함되어 있다.

3.1.1 규칙의 표현

규칙의 표현은 (그림 2)와 같다.

<Knowledge Base>::=<rule>
| <rule><Knowledge Base>
<rule> ::= **rule** <rule-name> (<certainty factor>)
<rule-class-name>
if <condition> **then** <action>
<rule-name> ::= <string>
<certainty factor> ::= 0.0 .... 1.0
<rule-class name> ::= <string>
<condition> ::= $\{<proposition>\}_1^7$
<action> ::= <proposition>
<proposition>::=<text><value-type>
{<operator> <value>}
<text> ::= <string>
<string> ::= {(<letter> | <digit>)}
<letter> ::= A| ... |Z|a|...|z
<digit> ::= 0|1|...|9
<operator>::='=' | '!=' | '<' | '<=' | '>' | '>='
<value>::=<string>| <number> | <function-name>
<number> ::= <integer> | <real>
<function-name> ::= <string>
<value_type> ::= t | f | s | n | r | x | i
[참고] t : true, f : false, s : string, n : integer,
r : real, x : function, i : interface of Database.

그림 2. 규칙의 표현
Fig. 2. Representation of rules

3.1.2 절차 부가 (Procedural Attachment)

생성 규칙 표현법이 갖고 있는 장점들 중 하나는 다양한 지식을 표현할 수 있다는 것이다. 지식기반 시스템은 필요한 경우 함수를 호출하여 비결정적인 값을 계산할 수 있어야 한다.

예를 들면, rule (certainty factor) <modeler>
if type of model, s=triangle
surface, x=calcul_surface_tri(10, 20)
surface, r >= 100.0
then triangle_model_no = 1

이 경우 함수는 0개 이상의 매개변수를 가진다.

추론 도중 이러한 지식을 만나면 먼저 함수 지식 인터프리터가 이 문장을 분석하여 함수명과 매개 변수의 개수 및 매개 변수를 분리한 뒤 절차 라이브러리에 있는 해당 함수를 진행시킨다.

3.1.3 데이터베이스의 지식에 대한 호출

지식기반 시스템에서 추론 엔진은 특정한 영역의 지식과 사실(fact)들을 가지고 추론을 행한다. 그러나 경우에 따라서는 사실의 양이 많아지고 데이터베이스에 저장되어 있기 때문에 데이터베이스 내에 저장된 사실들을 호출해야 할 경우가 있다.

따라서 본 시스템은 데이터 베이스내의 정보의 호출, 추가, 제거 및 수정할 수 있는 기능을 갖추었다. 관계(relation) 이름, 엔티티(entity)이름, 키(key)이름, 레코드(record)이름을 가지고 조작할 수 있다.

예를 들면,

rule2 (certainty factor) <modeler>
if eq1, i = a(modeler1, equation, pri_key1, edge1_id)
eq2, i = a(modeler1, equation, pri_key1, edge2_id)
calcul_state, x = calcul_parall(eq1, eq2)
calcul_state, s = intersection
then delete_edge, i = s(relation_name, key_col, edge2_id)

이것은 절차 부가의 일종이므로 수행과정은 절차 부가와 동일하다.

3.2 지식 컴파일러

지식 베이스에 저장된 규칙은 지식 컴파일러에 의해 네트워크 방식의 내부 표현 구조로 나타내어진다. 이는 추론 도중 가능성이 있는 규칙만을 탐색함으로써 추론 엔진의 추론 속도를 증가시키려는 데 그 목적이 있다. 또한 네트워크로 표현함으로써 도달될 수 없는 규칙, 더 이상 연결되지 않는 규칙들을 쉽게 찾아낼 수 있다는 장점을 가지기 때문이다. 지식은 오류를 점검하는 검증 과정을 거쳐 오류가 없으면 네트워크 구조에 추가되며 최종적으로 지식베이스에 저장된다.

3.3 지식 검증 시스템

지식 공학자나 특정 분야 전문가 즉 사용자가 새로운 지식을 지식베이스에 추가하고자 할 경우, 지식 검증 시스템은 지식이 추가되기 전에 정확성을 검증한다. 이 시스템에서는 확실한 특성의 리스트와 가능한 특성의 리스트가 생성되고 이들을 이용하여 지식이 첨가되기 전 중복, 모순, 순환 및 도달될 수 없는 규칙과 더 이상 연결되지 않는 규칙 등의 오류가 검사된다. 만약 오류가 있다면 지식 제공자의 승인을 거쳐 오류를 제거한 뒤 새로운 규칙이 삽입된다.

3.3.1 확실한 특성의 리스트와 가능한 특성의 리스트

각 프레미스의 확실한 특성과 가능한 특성은 지식 획득 시스템에 의해 구해진 후 리스트 구조로 표현된다. 이 때 확실한 특성에는 상식과 범용 지식이 포함된다.

- 정의 1) 네트워크의 규칙을 Ri, i=1..n으로 표시하고, 하나의 규칙이 조건 부분(C)과 실행 부분(A)으로 표현되면 규칙 Ri의 조건부분의 모든 프레미스는 SET Ri라 정의한다.
- 정의 2) R1 → R2 → ⋯ → Rn이고 Rn의 실행 부분이 A인 일련의 연속된 규칙들을 A의 경로라 하며 T(A)라 정의한다. 만약 A가 조건으로만 사용된다면 A의 경로가 없다.
- 정의 3) 만약 E의 실행이 P의 실행을 내포하면 P는 E의 확실한 특성이라 정의한다. 표현 방법은 E → P 이다. 다음 3조건 중 하나를 만족하면 P는 E의 확실한 특성이다
: Certain(E) = {P| P는 E의 확실한특성이다}.

① $\forall K \in [1..q],\ \exists i \in [1..n],\ Ri \in Tk(E)$ and $P \in$ Set Ri.

즉 $Certain(E) = \bigcap_{i=1}^{q} Antecedent(R_{ii})$

$$Antecedent(R_i) = \bigcup_{j=1}^{m} Certain(P_{ji}) \cup \{P_{1i}, P_{2i}, \ldots P_{mi}\}$$

$Certain(P_{ji}) = \varnothing$: 만약 Pji에 관련된 경로가 없다면.

② 실행 부분이 P인 규칙이 존재해서 그 규칙의 모든 조건 부분이 E의 확실한 특성일 경우.

③ 하나의 조건 not(P)와 실행부 A를 가지는 규칙이 존재하고 not(A)가 E의 확실한 특성일 경우.

• 정의 4) 가능한 특성은 일어날 수도 있고 그렇지 않을 수도 있는 특성을 말한다. 만약 P의 실행이 E의 실행에 기여한다면, P는 E의 가능한 특성이라 정의한다. 만약 아래의 조건 중 하나를 만족하면 P는 E의 가능한 특성이다 : Eventual (E) ={P| P는 E의 보조의 특성이다}.

① $\exists K \in [1...q]$, $\exists i \in [1...n]$, $Ri \in Tk(E)$ and $P \in$ Set Ri

즉 $Eventual(E) = \bigcup_{i=1}^{q} Antecedent(R_i)$

$$Antecedent(R_i) = \bigcup_{j=1}^{m} Eventual(R_{ji}) \cup \{P_{1i}, P_{2i}, \cdots P_{mi}\}$$

$Eventual(P_{ji}) = \varnothing$: 만약 Pji에 관계된 경로가 없다면.

② 실행 부분이 P인 규칙이 존재해서 그 규칙의 모든 조건 부분이 E의 가능한 특성일 경우.

③ 하나의 조건 not(P)와 실행부 A를 가지는 규칙이 존재하고 not(A)가 E의 가능한 특성일 경우.

결국 다음과 같은 결과가 얻어진다.

Certain(A) ⊂ Eventual(A).

3.3.2 지식베이스의 오류

본 논문에서는 지식의 중복, 모순, 순환, 도달되지 않는 규칙 그리고 더 이상 연결되지 않는 규칙을 오류라 칭한다.

(1) 중복

지식의 중복은 다음과 같이 점검된다.

• 구조적 중복과 수정
- Set R1 = Set R2 and A1 = A2 : 하나의 규칙은 제거한다.
- Set R1 ⊂ Set R2 and A1 = A2 : R1이나 R2 둘 중 하나의 규칙만 선택하는데 R2를 선택하는 것이 바람직하다.
- A1 ∈ Set R2 and #(Set R1 ∩ Set R2) >= 1 : Set R2 = Set R2 - Set R1

• 의미적 중복과 수정
- 하나의 규칙 안에서 조건 부분에 프레미스 X를 첨가하려는데 X의 확실한 특성 중 하나가 이미 소개되어진 프레미스 P인 경우 프레미스 P를 제거할 수 있다. 그렇지 않으면 상황을 열거한다.
- Set R1 = Set R2 이거나 Set R1 ⊂ Set R2 이고 A1 과 A2중 어느 하나가 다른 하나의 확실한 특성 또는 가능한 특성의 관계에 있을 때, 확실한 특성 또는 가능한 특성을 실행부로 가진 규칙을 제거한다.

(2) 모순

• 구조적 모순과 수정
- Set R1 = Set R2 and A1 ≠ A2 : 만약 A1 과 A2가 논리적 모순 관계이면 규칙 중 하나를 버려야 한다. 이 경우 전문가에게 수정의 메시지를 주어 Set R1 혹은 Set R2를 수정하거나 두 규칙 중 하나를 버린다. 단, 논리적 모순 관계란 해당되는 프레미스의 반의어거나 반의어의 가능한 특성 혹은 확실한 특성을 말한다.
- Set R1 ⊂ Set R2 and A1 ≠ A2 : 만약 A1과 A2가 논리적 모순관계이면 전문가의 지시대로 규칙 중 하나는 버려야 한다. 그렇지 않을 경우 규칙 R1은 그대로 두고 규칙 R2를 수정하여 첨가한다. 즉 Set R2 = (Set R2 - Set R1) ∪ {A1}.
- A1 = A2 and Set R1 ≠ Set R2 : 확실한 특성과 가능한 특성을 사용하여 Set R1과 Set R2의 프레미스들이 서로 모순관계에 있는지 검사한 후 모순관계가 아니면 첨가하고 모순관계이면 지식제공자에게 상황을 알리고 수정을 요구한다.

• 의미적 모순과 수정
- 하나의 규칙 안에서 이미 소개되어진 프레미스 P의 확실한 특성과는 반대 의미로서 묘사되어진 프레미스 X를 첨가하려 할 때 시스템은 모순의 근원을 알려주어 수정하도록 한다. 만일 X가 확증되었다면 프레미스는 기록되고 X가 P의 확실한 특성이 아니다라는 사실이 기억된다.
- 하나의 규칙 안에서 조건 부분에 프레미스 X를 첨가하려는데 X가 실행부분 A의 확실한 특성이나 가능한 특성과 반대의 의미를 가진 경우 프레미스 X를 첨가하면 모순이 발생함을 알린다.

(3) 순환

하나의 규칙 내에 프레미스 P를 첨가하려는데 실행부 A가 P의 가능한 특성 중 하나인 경우, 시스템은 사용자에게 경고하고 프레미스를 첨가하지 않고 지식 제공자가 선택하도록 한다.

(4) 더 이상 연결되지 않는 규칙

규칙의 실행부가 목표가 아니고 다른 규칙의 조건 부분에 포함되어 있지도 않다면, 그 규칙은 더 이상 연결되지 않는 규칙이다. 이 규칙은 지식베이스에 포함되지 말아야 하거나, 추론할 때 이 규칙의 실행부를 사용하는 다른 규칙이 빠진 경우이므로 지식 제공자에 의해 빠진 규칙이 보완되어야 한다. 네트워크 구조에서 다른 규칙으로의 링크가 없는 경우이므로 쉽게 찾아질 수 있다. (그림 3)에서 R7은 더 이상 연결되지 않는 규칙이다. P59는 목표가 아니면서 진출차수가 0이기 때문이다.

(5) 도달될 수 없는 규칙

한 규칙의 조건부가 입력 변수가 아니고 다른 규칙의 실행부도 아닐 때, 즉 규칙의 진입 차수가 0일 때 이 규칙의 실행부는 도달될 수 없는 절이다. 이 규칙은 필요 없는 규칙이거나 이 규칙의 조건 부분에 도달되기 위한 규칙이 빠진 경우이기 때문에 빠진 규칙이 보완되어야 한다. (그림 3)에서 R6은 도달될 수 없는 규칙이다. 왜냐하면 P57은 입력 변수가 아니고 R6로의 진입차수가 0이기 때문이다.

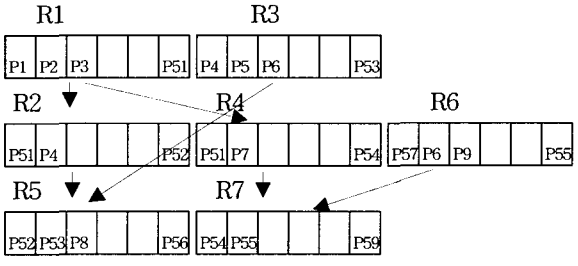

그림 3. 지식의 네트워크 표현의 예

Fig. 3. Example of Network Representation for Knowledges

4. 구현한 시스템 및 사례

4.1 지식 검증 시스템의 알고리즘

4.1.1지식의 첨가 알고리즘

구현한 시스템은 첨가하려는 규칙의 조건부분을 기존의 규칙을 토대로 확장하고 확실한 특성의 리스트와 가능한 특성의 리스트를 이용하여 규칙 내에서의 모순, 순환 등을 점검한다. 그런데 일반적인 쌍 검증 방식에서는 새로운 규칙을 첨가할 때 이 규칙이 영향을 미치는 모든 규칙과의 일관성 여부를 알아보기 위해 전방추론이나 후방추론을 한다. 본 시스템에서는 이러한 전방추론과 후방추론을 생략할 수 있도록 함으로써 시간이 많이 걸리는 단점을 보완하며 일관성을 유지하도록 하였다.

새로운 지식을 첨가하기위한 절차는 (알고리즘 1)에 있다.

```
function InsertRule (Rule R)
 begin                              // 지식의 오류점검
 NumverofRule ← 지식베이스내의 지식의 수
 for (i←1; i <= NumberofRule; i++)        // 1단계
    if (Set Ri ⊂ Set R and Ai ∉Set R)
            Set R ← Set R ∪ {Ai}

 NumberofCond ← #(Set Ri)                 // 2단계
 for (i←1; i <= NumberofCond; i++)
   if (Set R의 i번째 프레미스와 Set R의 나머지 프
       레미스의 관계가 모순이나 순환이면) 끝(오류).

 for (i←1; i <= NumberofCond; i++)        // 3단계
    Set R'←Set R- ∪(i번째 프레미스의 가능한 프레미스)
                 j∈ 가능한 특정의 리스트
 for (i←1; i <=NumberofRule; i++)         // 4단계
   if (Set R' = Set Ri)
         if (A = Ai)  끝(중복).
         else if (A와 Ai의 관계가 모순) 끝(모순).
 for (i←1; i <= NumberofRule; i++)        // 5단계
   if (A = Ai)
   begin
    set T ← Set Ri ∪ Set R'
    NumberofCond ← #(Set T)
    for (i←1; i <= NumberofCond; i++)
      if (Set R의 i번째 프레미스와 Set R의 나머지 프레
          미스의 관계가 모순이면)  끝(모순).
   end
                                    // 기존 규칙의 수정
 for (i←1; i <= NumberofRule; i++)        // 6단계
   begin
    if (A ∈ Set Ri)
      begin
      Set R and  Set Ri의 모순 검사
      if (모순관계이면)  끝(모순)
      end
    if (Set R ⊂ Set Ri)
      begin
      (Set Ri ← (Set Ri - Set R ) ∪  A
      규칙 Ri의 진입차수, 진출차수 수정
      end
   end

 for (i←1; i <= NumberofRule; i++)        // 7단계
  if (A ∈ Set Ri) 규칙 R의 진출차수에 1증가 &
                  규칙 Ri의 진입차수에 1증가
 for (i←1; i <= NumberofRule; i++)        // 8단계
    begin
      if (진입차수 = 0) 도달될 수 없는 절
      if (진출차수 = 0) 더 이상 연결되지 않는 절
    end
 지식베이스에 새로운 규칙 R 첨가          // 규칙첨가
end
```

(알고리즘1) 새로운 지식의 첨가

4.1.2 규칙의 제거 알고리즘

규칙의 제거는 규칙 R은 물론 R의 선행자의 링크까지 제거되는 것이다. 이때 도달될 수 없는 규칙과 더 이상 연결되지 않는 규칙이 나타나면 전문가에게 알려 수정하도록 한다.

```
function DeleteRule (Rule R)
 begin                              // 지식의 오류점검
   NumverofLink ← R의 진출 차수
   for (i←1; i<=NumberofLink; i++) //진출링크 제거
    begin
     if (Ri의 조건부분에서 R의 진입링크의
        삭제가 불가능한가) 오류(R의 삭제불가능)
     각 Ri의 진입차수를 1감소.
    end

 NumverofRule ← 지식베이스내의 지식의 수
   for (i←1; i<=NumberofRule; i++) // 진입링크수정
    if (R ∈ Set Ri의 진출 링크) 링크제거

   for(i←1; i <= NumberofRule; i++)
    begin
    if (진입차수 = 0) 도달될 수 없는 절
    if (진출차수 = 0) 더 이상 연결되지 않는 절
    end
 end
```

(알고리즘 2) 지식의 제거

4.2 예

지식베이스에 다음과 같이 6개의 지식이 있다.

R1 : P1 , P2 , P3 → P51
R2 : P51 , P4 → P52
R3 : P4 , P5 , P6 → P53
R4 : P51 , P7 → P54
R5 : P52 , P8 → P54
R6 : P51 , P5 , P9 → P55

다음과 같이 차례로 규칙을 첨가해본다.

① R7 : P51 , P10 →P56 : 수정 없이 첨가된다.
② R8 : P1 , P2, P3 , P9 → P57 : R8은 다음과 같이 변경되어 첨가되고 R6도 변경된다.
   R8 : P51, P9 → P57
   R6 : P57 , P5 → P55
③ R9 : P51 , P4 , P10 → P58 : R9는 다음과 같이 변경되어 첨가된다.
   R9 : P52 , P10 → P58
④ R10 : P52 , P11 → P51 : 순환이라 명시하며 첨가되지 않는다.
⑤ R11 : P52 , ~P4 → P59 : P52의 가능한 특성으로 P4가 존재하므로 P4와 ~P4는 서로 모순 관계이므로 모순이라 지적하고 첨가되지 않는다..
⑥ R12 : ~P12 → P60 : 수정 없이 첨가된다.
⑦ R13 : P12 , P60 , P13 → P61 : 조건절에 P12 와 P60의 확실한 특성인 ~P12가 있는 경우이므로 모순임을 알리며 첨가되지 않는다.
⑧ R14 : P1 , P2 , P3 → P62 : R1과 R14의 조건절이 같아서 P51과 P62가 모순 관계인지를 먼저 확실한 특성의 리스트와 가능한 특성의 리스트를 사용하여 점검한 뒤 모순이 발견되면 모순 관계임을 알리며 모순이 발견되지 않을 경우에는 지식 제공자에게 문의하여 모순이 아닌지를 확인한

후 다음과 같이 수정되어 저장된다.

R14 : P51 → P62로 저장된다.

⑨ R15 : P1,~P8 → P51 : P8과 P1의 관계가 모순관계는 아니지만, R15를 삽입하면 첨가과 정의 단계 6에서 R5가 모순이 된다. R5 는 P52, P8 → P54이므로 확장시켜 보면 P1, P2, P3, ~P8, P4, P8 → P54규칙으로 발견되므로 시스템은 지식의 제공자에게 R15을 첨가할 경우 R5가 모순으로 변함을 알리고 R15를 삽입할 것인지를 묻는다.

위 예의 각 과정에서 더 이상 연결되지 않는 규칙과 도달될 수 없는 규칙의 리스트를 보여준다.

5. 시뮬레이션 분석

이 논문에서 제시한 지식검증 시스템의 기본 알고리즘의 시간 복잡도에 영향을 미치는 요인은 다음과 같다.

n : 지식베이스내의 규칙의 수

l : 확실한 특성의 리스트의 수

m : 가능한 특성의 리스트의 수

k : 하나의 규칙 내에서 조건부의 최대 수

h : 하나의 리스트에서 항의 최대 수

다음은 최악의 경우를 고려하여 분석해 본다. 즉 새로운 규칙이 모든 지식에 영향을 미친다고 가정하자.

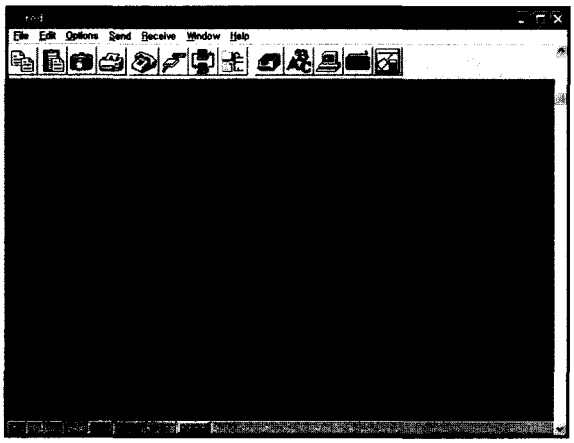

그림 4. 예 : 지식 검증 시스템의 실행

Fig. 4. Examples : Execution of Knowledge Verification System

이때 각 단계의 비교 횟수는 다음과 같다.

1단계 : O(n · k)

2단계 : O(k2 · h)

3단계 : O(k · h)

4단계 : O(n · k)

5단계 : O(n) + O(k2 · h)

6단계 : O(n) + O(k2 · h) + O(n · k)

7단계 : O(n · k)이다.

8단계 : O(n)이다.

따라서 8단계까지 각 단계를 모두 합한 비교 횟수는 $O(4nk + 3k2h + kh + 3n) = O(n(4k+3) + kh(3k+1)) \approx O(nk+k2h)$이며 k 와 h는 보통 작은 상수이다. 그러나 기존의 쌍 검증 방식에서는 일관성 검증을 위해 전방추론 혹은 후방추론을 해야 하므로 최악의 경우 지수승의 특성을 보인다[2].

이와 같이 지수승의 성능을 보이는 NP-복잡성 문제인 일관성 검증 문제를 본 시스템은 O(nk+k2h)정도의 효율적인 검증 절차를 거쳐 오류가 없을 경우에만 지식의 삽입과 제거 연산을 수행한다.

6. 결 론

특정 영역의 지식과 절차적 지식 및 데이터베이스의 선언적 지식 등 다양한 종류의 지식을 표현할 수 있는 생성 규칙의 표현 방식은 널리 사용되고 있다. 이러한 지식들은 추론을 수행하기 위해 사용되어지는데 추론 결과의 신뢰도를 높이기 위해서 지식의 일관성 검증을 위한 지식 검증 시스템이 제안되고 구축되었다. 지식 검증 방식 중 기존의 쌍 검증 방식은 규칙의 수가 많을 경우 시간이 많이 걸리고 중요한 규칙이 생략될 수 있는 단점이 있다. 따라서 본 논문에서는 이를 보완하기 위해 확실한 특성의 리스트와 가능한 특성의 리스트를 사용하고 검증 방법 및 단계를 보완하여 개선된 쌍 검증 방식을 기반으로 한 지식 검증 시스템을 구축하였다. 개선된 쌍 검증 방법은 확실한 특성과 가능한 특성의 리스트를 사용하여 오류를 검색할 때 전방 추론이나 후방 추론을 해야 하는 과정의 대부분을 생략할 수 있게 함으로써 시간이 많이 걸리는 단점을 보완함과 동시에 오류 점검 단계에서 모든 가능성을 고려함으로써 생략되는 규칙이 없어 정확한 오류 점검을 수행한다. 특히 확실한 특성의 리스트에는 일반적인 지식이나 상식 등도 포함될 수 있어서 의미적 오류도 찾아 낼 수 있다. 따라서 개선된 쌍 검증 방식을 이용한 지식 검증 시스템에서 새로 첨가될 지식은 기존의 지식과의 중복, 모순, 순환, 도달되지 않는 규칙과 더 이상 연결되지 않는 규칙 등의 오류를 구조적 및 의미적 측면에서 판별한 뒤 오류가 없는 경우 지식베이스에 첨가된다.

결국 본 논문에서는 지식 기반 시스템의 추론 결과의 신뢰도가 전적으로 지식 재공자에 의해 결정되는 방식을 피하고 지식이 첨가되기 전에 지식 검증 시스템 내에서 지식의 오류를 자동적으로 찾아내어 지적하고 수정하거나 제거한 후 첨가되게 함으로써 추론 결과의 신뢰도가 높아지고 지식 제공자가 보다 쉽게 지식을 첨가 및 저장할 수 있도록 하였다.

본 지식 검증 시스템은 0+ 차원의 생성 규칙을 기반으로 하는 지식 기반 시스템의 필수적인 요소가 될 것이며 기계학습 분야에도 적용가능하다. 앞으로 본 시스템을 결합한 특정 분야의 에이전트를 구축하는 것이 향후 연구 과제이다.

참 고 문 헌

[1] N.Botten, A.Kusiak and T.Raz, "Knowledge Bases : Integration, Verification and Partitioning", European j. Operational Research, vol 42, pp.111-128, 1989.

[2] Laure Brisoux, Eric Gregoire, Lakhdar Sais, "Checking Depth-Limited Consistency and Inconsistency in Knowledge-Based Systems", International Journal of Intelligent Systems, vol. 16, 2001, pp 319-331.

[3] B.Cragun and H.Steudel, "A Decision-Table Based Processor for Checking Completeness and Consistency in Rule-Based Expert Systems", Int'l j. Man-Machine Studies, vol 26, no. 5, pp.633-648, 1987.

[4] Walling R. Cyre, "Capture, integration, and analysis of digital system requirements with conceptual graphs", IEEE Trans. on Knowledge and data engineering, vol 9, no. 1, pp 8-23, jan-feb, 1997.
[5] Christoph F. Eick, "Rule-Based Consistency Enforcement for Knowledge-Based systems", IEEE Trans. on Knowledge and data engineering, vol 5, no.1 ,pp4-13, feb, 1993.
[6] Gerard Ellis, "Compiling Conceptual Graphs", IEEE trans. on Knowledge and data engineering, vol 7, no 1, pp. 68-81, feb, 1995.
[7] P. LeBeux, D. Fontaine, "Un systeme d'acquisition des connaissances pour systemes experts", Technique et Science Informatique, vol 5, no.1, pp7-20, 1986.
[8] N.K.Liu, "Formal Verification of Some Potential Contradictions in Knowledge Base Usion a High Level Net Approach", Applied Intelligence, vol.6, no. 4, Oct, 1996
[9] B.Mazure, L.Sais, E.Gregoire, "Boosting complete technique thanks to local search methods", Annuals of mathematics and artificial intelligence, Kluwer, pp 319-331, 1998
[10] P.Morizet-Mahoudeaux, "Maintaining Consistency of Database During Monitoring of an Evolving Process by a Knowledge-Based System", IEEE Trans. on systems, man and cybernetics, vol 21, no. 1, pp 47-60, 1991.
[11] D.L.Nazareth, "Issues in the Verification of Knowledge in Rule-Based Systems", I.J.Man-Machine Studies, vol. 30, pp. 255-271, 1989.
[12] D.L.Nazareth and M.H.Kennedy, "Verification of Rule-Based Knowledge Using Directed Graphs", Knowledge Acquisition, vol.3, pp.339-360, 1991.
[13] D.L.Nazareth, "Investigating the Applicability of Petri Nets for Rule-Based System Verification", IEEE Trans. on Knowledge and data engineering, vol 5, no. 3, pp.402-415, 1993.
[14] T.A. Nguyen, N.A.Perkins, T.J.Laffey and D.Pecora, "Checking an Expert system knowledge base for consistency and completeness", Proc. IJCAI85, LA, Califonia, pp. 375-378, 1985.
[15] T.A.Nguyen, W.A.Perkins, T.J.Laffey and D.Pecora, "Knowledge Based Verification", AI Magazine, vol 8, no 2, pp.69-75, 1987.
[16] M. Ramaswamy, S. Sarkar, Ye-Sho Chen, "Using directed hypergraphs to verify rule-based expert systems", IEEE Trans. on Knowledge and data engineering, vol.9, no. 2, march-april, pp. 221-237, 1997.
[17] B. Selman, H.Kautz, D.McAllester, "Computational challenges in propositional resoning and search", Proc. IJCAI97, 1997
[18] G.Valiente, "Verification of knowledge based redundancy and subsumption using graph transformations", Int'l J. Expert System, vol.6, no.3, pp341-355, 1993.

저 자 소 개

서의현

1980년 : 이화여자대학교 수학과 이학사
1980년~1982년 : 한국개발연구원, 연구원
1985년 : 프랑스 꽁삐엔느 대학교 컴퓨터공학과 (공학석사)
1988년 : 프랑스 꽁삐엔느대학교, 컴퓨터공학과(공학박사)
1990년~현재 : 목원대학교 컴퓨터공학과 교수

관심분야 : 지식공학, HCI, 기계학습, 에이전트 시스템.
e-mail : ehsuh@mokwon.ac.kr