지식의 일관성과 완결성을 위한 구조적 및 의미론적 검증

서 의 현†

요 약

규칙 기반 지식표현은 전문가 시스템의 지식을 저장하고 조작하는 기법중 가장 많이 사용되는 기법이다. 그런데 이 방법으로 표현된 지식의 양은 점점 많아지며 그 관계가 복잡해지고 내용이 변경될 수 있기 때문에 지식베이스의 일관성 및 완결성을 유지하는 검증 시스템이 필요하다. 따라서 본 논문에서는 새로운 규칙이 첨가될 때 야기될 수 있는 오류를 구조적 측면과 의미론적 측면에서 검증한 후 자체 수정하거나 자체 수정이 불가능할 경우 전문가가 수정하도록 하여 일관성과 완결성이 확증된 후 새로운 규칙이 추가되어질 수 있도록 함으로써 전문가 시스템의 신뢰도를 높이고 효율성을 제고할 수 있는 검증시스템이 제안되었다.

Structural and Semantic Verification for Consistency and Completeness of Knowledge

Euy-Hyun Suh†

ABSTRACT

Rule-based knowledge representation is the most popular technique for storage and manipulation of domain knowledge in expert system. By the way, the amount of knowledge increases more and more in this representation technique, its relationship becomes complex, and even its contents can be modified. This is the reason why rule-based knowledge representation technique requires a verification system which can maintain consistency and completeness of knowledge base. This paper is to propose a verification system for consistency and completeness of knowledge base to promote the efficiency and reliability of expert system. After verifying the potential errors both in structure and in semantics whenever a new rule is added, this system renders knowledge base consistent and complete by correcting them automatically or by making expert correct them if it fails.

1. 서 론

전문가 시스템에서 지식을 저장하고 조작하는 기법 중 가장 잘 알려진 것은 생성 규칙이다. 이 방법은 규칙들을 모듈화하여 나타낼 수 있고 규칙들로서 저장된 지식을 비절차적 방식으로도 사용할 수 있으며 여러 종류의 지식을 쉽게 표현할 수 있다는 장점 때문이다. 그러나 이 표현 방식에서는 문제점도 있다. 지식의 양이 방대한 경우가 많기 때문에 프로그램 개발자나 전문가가 내용 전체를 보기가 어렵고, 전문가는 때때로 직관적으로 생각하므로 추론에서의 많은 과정을 생략한 채 지식을 지식베이스에 삽입할 수 있다는 것이다. 또한 지식베이스는 점진적인 방식으로 오랜 기간에 걸쳐 구

† 정 회 원 : 목원대학교 컴퓨터공학과 교수
논문접수 : 1997년 10월 27일, 심사완료 : 1998년 5월 29일

[illegible] 여러 명의 전문가가 지식을 함께 저장하므로 지식의 중복이나 순환의 문제를 야기 시킬 수 있다. 이러한 단점들은 구조적 및 의미론적 관점에서 일관성(consistency)과 완전성(completeness) 문제로 집약된다[1]. 일관성과 완전성 문제를 해결하기 위해서는 전문가 시스템이 지식베이스를 사용하기 전에 정확성을 검증하고 찾아진 오류를 자체 수정하며 자체 수정이 불가능한 경우 전문가에게 문의하여 수정하는 시스템이 요구된다.

이러한 지식베이스의 검증을 위한 많은 시도가 있었다. 지식 경영 시스템(Knowledge Base Management System :KBMS) 내에 일관성 제약 조건을 제시하고, 제약 조건을 위반 했을 경우 예외 처리를 실행하기 위하여 실행 형태 제어 규칙(Activation Pattern Controlled Rule)을 사용한 방식이 있다[4]. 그런데 이 방식은 부분적인 일관성만을 자동으로 처리할 수 있을 뿐이다.

의존 관계표를 사용하여 일관성과 완전성을 점검하는 방법[12]은 역시 방식으로 처리하기 때문에 시간이 많이 걸리는 단점이 있다.

결정표 (decision table)를 사용하는 방식[2]에서는 간혹 오류의 존재만을 발견하고 오류의 위치를 정확하게 찾아내지 못하는 단점이 있다. 또한 복잡한 관계를 가지는 규칙들을 추론 과정 중 연결된 규칙이 많아 지수 함수적인 성능을 나타낸다[9].

쌍 검증(pairwise checking) 방식에서는 생성된 쌍들이 적용 가능한 규칙들과 비교된다[6][8]. 이 때 경우에 따라서는 의미 없는 규칙은 취급하고 가치 있는 규칙은 생략하는 일도 발생할 수 있으며 규칙의 수가 많아질 때 시간이 많이 걸리는 단점이 있다[7].

그래프 표현 방식은 프레미스의 개념적인 의존 관계를 나타내기 쉬운 표현법으로서 방향 그래프 (directed graph)[10], 추론 그래프 (inference graph)[13], 하이퍼 그래프(hypergraph)[15]등의 많은 시도가 있었다. 그러나 이러한 방법들은 복합절과 단순절 사이의 의존 관계를 나타내기 어렵다는 단점이 있다[14]. 따라서 그래프 표현 방식은 노드를 첨가하거나 전이를 묘사하거나 혹은 하이퍼 아크를 사용하여 복합절을 묘사함으로써 중복과 포함을 정확히 발견하여 수정한다. 그렇지만 모순과 불완전성은 여전히 완벽하게 처리되지 못하는 단점이 있다. 방향 하이퍼 그래프 (directed hypergraph)[14]는 이러한 그래프의 단점을 보완하였지만 고려되어져야 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 구조로서 지식을 체계적으로 분류하여 계층적으로 저장하고 편집 방법을 통해 중복을 제거한다[3][5]. 그러나 이 방식은 표현의 다양성 때문에 여러 곳에 있는 비일관성 오류를 점검하기 어렵고 극히 제한적인 일관성만을 검진할 수 있다.

페트리 네트(Petri Net)를 이용한 방식[7][11]은 일관성 검진이 정해진 실행 순서로 진행되어야 하는데 형식이 완전하지 못하면 제약 조건이 적용되지 못하고 수행시간도 많이 걸리는 단점이 있다.

따라서 본 논문은 확실한 특성의 리스트와 우발적 특성의 리스트를 첨가하고 검증 방법 및 단계를 개선하는 형태로 쌍 검증 방식을 보완한다. 이 리스트들과 개선된 검증 방법은 오류 검색 시 전방 추론이나 후방 추론을 해야 하는 과정의 대부분을 생략할 수 있게 해 줌으로써 시간이 많이 걸리는 단점을 보완해 주고 오류 점검 단계에서 모든 가능성을 고려함으로써 생략되는 규칙이 없어 정확한 오류 점검이 가능하다. 특히 확실한 특성의 리스트에는 일반적인 지식이나 상식 등도 포함될 수 있어서 의미론적 오류도 찾아 낼 수 있다. 그래서 본 논문은 부정의 프레미스를 포함한 네트워크로 지식을 표현하고 보완된 쌍 검증 방식을 이용하여 구조적 측면에서뿐만 아니라 의미론적 측면에서도 중복, 모순, 순환의 일관성 오류는 물론 도달될 수 없는 절과 더 이상 연결되지 않는 절등의 완결성 오류를 찾아내어 자체 수정하거나 자체 수정이 불가능할 경우 전문가에게 수정하도록 메시지를 전달함으로써 수정하는 방법을 제시한다.

2. 지식의 표현

2.1. 지식베이스의 구조

지식베이스의 구조는 생성 규칙이다. 이것은 규칙들을 모듈화하여 나타낼 수 있고 비절차적 방식으로도 규칙들로서 저장된 지식을 사용할 수 있을 뿐 아니라 여러 종류의 지식을 쉽게 표현할 수 있다는 장점 때문이다. 지식의 표현은 조건 부분과 실행 부분으로 나타낸다 : if P_1 and P_2 and P_3 then A_1. 우리는 규칙을 동일된 형태로 나타내기 위해 다음과 같은 제한된 형태

에 생성 규칙을 사용한다. 규칙의 조건 부분은 여러 연산자(and)를 사용하여 여러 개의 프레미스로 나타낼 수 있고 실행 부분은 단 하나의 프레미스로만 표현될 수 있다. 따라서 실행부분이 여러 개의 프레미스로 된 규칙은 여러 개의 규칙들로 나뉘어진다. 즉

if P_1 and P_2 and P_3 ---> A_1 and A_2 이면,

=> if P_1 and P_2 and P_3 ---> A_1

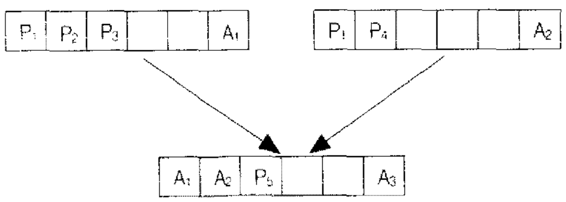
if P_1 and P_2 and P_3 ---> A_2 의 두 개의 규칙으로 표현한다. 또한 부정의 프레미스(~P)도 표현할 수 있다.

2.2. 지식의 내부적 표현

지식은 네트워크와 리스트 구조로 구성된다. 규칙은 네트워크 구조로 나타내어지고 프레미스들의 확실한 특성과 우발적 특성은 리스트 구조로 나타내어진다.

(1) 네트워크 구조

규칙의 내부적 표현 방식으로 네트워크 구조를 사용한다. 이는 추론 도중 가능성이 있는 규칙만을 탐색함으로써 추론 엔진의 추론 속도를 증가시키려는데 그 목적이 있다. 또한 네트워크로 표현함으로써 도달될 수 없는 점, 더 이상 연결되지 않는 점들을 쉽게 찾아낼 수 있다는 장점을 가지기 때문이다. 지식은 오류를 검검하는 편집 과정을 거쳐 오류가 없으면 네트워크 구조에 보존된다. 지식의 내부적 표현 방법은 다음과 같다.



(그림 1) 네트워크 구조
(Fig. 1) Network structure

(2) 리스트 구조 : 확실한 특성의 리스트, 우발적 특성의 리스트

각 프레미스의 확실한 특성과 우발적 특성을 찾아 리스트 구조로 지식 베이스에 저장한다. 이 때 확실한 특성에는 상식과 범용 지식이 포함된다.

정의 1) 네트워크의 규칙을 R_i, i=1..n 으로 표시하고, 하나의 규칙이 조건 부분(P)과 실행 부분(E)으로 유일하면 규칙 R의 조건 부분과 [illegible] Set R_i라 정의한다.

정의 2) R_1 -> R_2 -> ... -> R_n이고 R_n의 실행 부분이 E인 일련의 연속된 규칙들을 E의 경로라 하며 T(E)라 정의한다. 만약 E가 조건으로만 사용된다면 E의 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 확실한 특성이다}.

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

1을 알고리즘으로 표시해보면 다음과 같다.

$$Certain(A) = \bigcap_{i=1}^{q} Antecedent(R_i)$$

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

$Certain(P_{ij})$ = 0 만약 P_{ij}에 관련된 경로가 없다면.

2는 전방 추론에 해당되며 3은 후방 추론에 해당된다.

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

1. $\exists K \in [1...q]$, $\exists i \in [1...n]$, $R_i \in T_k(E)$ and $P \in$ Set R_i
2. 실행 부분이 P인 규칙이 존재해서 그 규칙의 모든 조건 부분이 E의 우발적 특성일 경우.
3. 하나의 조건 not(P)와 실행부 A를 가지는 규칙이 존재하고 not(A)가 E의 우발적 특성일 경우.

주. T(E)에 속하는 규칙들의 모든 조건들을 말한다. 1의 알고리즘은 다음과 같다.

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

$\text{Antecedent}(R_j) = \bigcup_{i=1}^{m} \text{Eventual}(P_{ji}) \cup \{P_{1i}, P_{2i}, \ldots, P_{mi}\}$

$\text{Eventual}(P_{ji}) = 0$: 만약 P_{ji}에 관계된 정보가 없다면. 결국 다음과 같은 결과가 얻어진다. $\text{Certain}(A) \subset \text{Eventual}(A)$.

예 1) R_1 : if P_1 and P_2 and P_3 - -> E_1이라면 SET R = { P_1, P_2, P_3 } 이다.

예 2) 확실한 특성의 정의 ②항의 예를 들면 다음과 같다.

A,B,C -> P
E -> A, B, C
===> E -> P

예 3) 확실한 특성의 ③항의 예는 다음과 같다.

not(P) -> A
E -> not(A)
===> E -> P

예 4) '그 동물은 긴 머리털을 가진다'(P) 는 '그 동물은 말이다'(E) 의 확실한 특성이다.

예 5) '동물은 네 발을 가진다'는 '동물은 포유류이다'의 우발적 특성이다.

예 6) 다음 규칙에서 x의 확실한 특성을 구해본다.

R_1 : a,b,c -> x
$0R_2$: a,d -> x
R_3 : b,e,f -> y
R_4 : g,h,f -> y
R_5 : a -> z 라 하자.

x의 확실한 특성을 구하기 위해

$\text{Certain}(x) = \text{Antecedent}(R_1) \cap \text{Antecedent}(R_2)$

$\text{Antecedent}(R_1) = \{a,b,c\} \cup \text{Certain}(a) \cup \text{Certain}(b) \cup \text{Certain}(c)$.

$\text{Certain}(a) = \{z\} \cup \text{Certain}(z) = \{z\}$

$\text{Certain}(b) = \emptyset$

$\text{Certain}(c) = \emptyset$

$\therefore \text{Antecedent}(R_1) = \{a,b,c,z\}$

$\text{Antecedent}(R_2) = \{a,d\} \cup \text{Certain}(a) \cup \text{Certain}(d) = \{a,d,z\}$

$\therefore \text{Certain}(x) = \{a,z\}$

따라서 x의 확실한 특성은 a와 z이다.

3. 지식베이스의 오류

본 논문에서는 지식의 중복, 모순, 순환, 도달되지 않는 점, 더 이상 연결되지 않는 점을 오류라 칭한다.

3.1. 중복

지식의 중복은 다음과 같이 점검된다.

(1) 구조적 중복과 수정

- Set R_1 = Set R_2 and A_1 = A_2. 이 경우 하나의 규칙은 제거한다.
- Set R_1 ⊂ Set R_2 and A_1 = A_2. 이 경우 R_1이나 R_2 중 하나의 규칙만 선택하는데 R_2를 선택하는 것이 바람직하다.
- A_1이 Set R_2안에 있고 #(Set R_1 ∩ Set R_2) >= 1 : Set R_2 = Set R_2 - Set R_1

(2) 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 R_1 = Set R_2 이거나 Set R_1 ⊂ Set R_2 이고 A_1과 A_2 중 어느 하나가 다른 하나의 확실한 특성 혹은 우발적 특성의 관계에 있을 때, 확실한 특성 혹은 우발적 특성을 실행부로 가진 규칙을 제거한다.

예 7) 식물은 외떡잎 식물이다(P)는 그 식물은 옥수수이다(X)의 확실한 특성이다.

그런데 이때 다음과 같은 규칙을 첨가하려 한다고 가정하자.

if 그 식물은 외떡잎 식물이다 and
if 그 식물은 옥수수이다
then ------

이때 이 식물은 외떡잎 식물이다는 중복된 프레미스이므로 제거할 수 있다.

3.2. 모순

(1)구조적 모순과 수정

- Set R_1 = Set R_2 and $A_1 \neq A_2$. 만약 A_1 과 A_2가 논리적 모순 관계이면 규칙 중 하나를 버려야 한다. 이 경우 전문가에게 수정의 메시지를 주어 Set R_1 혹은 Set R_2를 수정하거나 두 규칙 중 하나를 버린다. 단 논리적 모순 관계란 해당되는 프레미스의 반의어거나 반의어의 우발적 특성 혹은 확실한 특성

을 발한다.

■ Set $R_1 \subset$ Set R_2 and $A_1 \neq A_2$. 이 경우 A_1과 A_2가 논리적 모순관계이면 전문가의 지시대로 규칙 중 하나는 버려야 한다. 그렇지 않을 경우 규칙 R_1은 그대로 두고 규칙 R_2를 수정하여 첨가한다. 즉 Set R_2 = (Set R_2 - Set R_1) $\cup$ {A_1}.

예 8) R_1 : if 새끼를 낳으면 then 포유류이다.

R_2 : if 새끼를 낳고 두발로 걸으면 then 인간이다.

---〉 수정된 R_2': if 포유류이고 두발로 걸으면 then 인간이다.

(2) 의미적 모순과 수정

이미 소개되어진 프레미스 P의 확실한 특성과는 반대 의미로서 묘사되어진 프레미스 X를 첨가하려 할 때 시스템은 모순의 근원을 알려준다. 만일 X가 확증됐다면 프레미스는 기록되고 X가 P의 확실한 특성이 아니라는 사실이 기억된다.

예 9) if 육식동물 and ..... then 고양이과이다.

if 동물이 고양이과에 속하고 동물이 초식동물이면 then ----.

이 경우 시스템은 두개의 초기 프레미스들 사이에 모순이 발생함을 지적하고, 두 번째 프레미스는 제거된다.

3.3. 순환

하나의 규칙 내에 프레미스 P를 첨가하려는데 실행부 A가 P의 우발적 특성중 하나인 경우, 시스템은 사용자에게 경고하고 프레미스를 첨가하지 않고 규칙의 첨가자가 선택하도록 한다.

예 10) if 동물이 고양이과이다 and

---

then 동물은 포유동물이다.

그런데 이미 포유동물은 고양이과의 확실한 특성이다. 따라서 우발적 특성도 된다.

즉 'if 포유동물 and ..... then 고양이과이다'라는 규칙이 존재한다. 그러므로 순환된다.

3.4. 더 이상 연결되지 않는 절

규칙의 실행부가 목표가 아니고 다른 규칙의 조건 부분에 포함되어 있지도 않다면, 그 규칙은 더 이상 연결되지 않는 규칙이다. 이 규칙은 지식베이스에 포함되지 말아야 하거나 최종 목표를 추론할 때 사용되기 위해 이 규칙의 실행부를 사용하는 다른 규칙이 빠진 경우이다. 네트워크 구조에서 다른 규칙으로의 링크가 없는 경우이므로 쉽게 찾아질 수 있다.

3.5. 도달될 수 없는 절

한 규칙의 조건부가 입력 변수가 아니고 다른 규칙의 실행부도 아닐 때, 즉 규칙의 진입 차수가 0일 때 이 규칙의 실행부는 도달될 수 없는 절이다. 따라서 이 규칙은 필요없는 규칙이거나 이 규칙의 조건 부분에 도달되기 위한 어떤 규칙이 빠진 경우이다.

4. 새로운 지식의 첨가와 제거

4.1. 지식을 첨가하기 전의 오류점검

새로운 지식을 첨가하기 전에 오류의 점검 과정을 차례로 기술하면 다음과 같다.

단계 1 : 첫 단계는 지식베이스의 규칙들의 각각인 R_i와 첨가하려는 규칙 R을 비교하는 것이다. 모든 R_i에 대해 만약 Set $R_i \subset$ Set R, $A_i \notin$ Set R then Set R = Set R $\cup$ {A_i}

예 11) R_1 : P_1 and P_2 --〉 A_1

R_2 : P_2 and P_3 --〉 A_2

and

R : P_1 and P_2 and P_3 and P_4 —〉 A

---〉 R : P_1 and P_2 and A_1 and P_3 and A_2 and P_4 --〉 A

단계 2 : 첨가하려는 규칙 내부에서 조건 부분들 사이에 의미적 모순은 없는지 검증해본다.

: 지식의 의미적 모순 제거, 순환 제거. 이 때 조건부의 프레미스들을 서로 비교하여 모순관계를 점검하고 만약 모순이 발견되면 전문가에게 수정을 요구한다. 조건부와 실행부에 같은 프레미스가 있으면 순환이다.

단계 3 : 불필요한 프레미스만 남겨두고 SET R을 감축시킨다 : 지식의 구조적 및 의미적 중복 제거. Set R' = Set R - $\cup$ 우발적 프레미스(P_i)

단계 4 : 첨가하려는 R은 조건 부분들이 같은 규칙들과 비교된다 : 구조적 중복, 모순제거.

만약 하나의 규칙 R_i가 R과 같은 실행부를 가지면 R은 제거된다. 만약 R_i와 R의 실행 부분이 다르다면 시스템은 어느 것이 모순된 것인지, 혹은 둘 다 받아들일 것 인지를 검사한다.

단계 5 : 규칙 R은 실행 부분이 같은 규칙들과 비교된다 : 구조적 중복, 모순제거.
만약 조건 부분이 다른 R_i의 SET R_i 중 SET R 과 모순되는 것이 있으면 모순을 지적하고 전문가에게 수정을 요구한다.

4.2. 규칙의 첨가와 네트워크의 수정.

위의 5단계를 거쳐 오류를 제거하고 수정된 규칙은 이제 지식베이스에 첨가된다. 그런데 이 새로운 규칙이 첨가됨으로써 지식베이스의 규칙들을 더욱 간소화시킬 수 있는지를 점검해보아야 한다.

단계 6 : 새로 삽입될 규칙의 실행부가 다른 규칙의 조건부라면 새로운 규칙을 삽입함으로써 기존의 규칙들이 모순을 일으키지 않는지를 점검한다 : 구조적 및 의미적 모순 제거. 이 때 모순이 발견되면 전문가에게 알린다. 또한 첨가할 규칙 R의 Set R ⊂ Set R_i인 R_i의 조건부를 Set R_i =(Set R_i - Set R) ∪ {A}로 수정하고 진입 차수와 진출 차수도 수정한다 : 구조적 중복제거.

단계 7 : 새로 첨가할 규칙의 진입 차수와 진출 차수를 구한다.

단계 8 : 규칙의 진입 차수가 0인 규칙은 도달될 수 없는 절이고 더 이상 연결되지 않는 절들은 네트워크 구조에서 진출 링크가 없는 경우이다.

4.3. 규칙의제거

규칙의 제거는 R을 제거하고, R의 선행자의 링크를 없애며, R의 후속자의 링크를 없앤다. 이때 도달될 수 없는 절과 더 이상 연결되지 않는 절이 나타나면 전문가에게 알려 수정하도록 한다.

4.4. 예

R_1 : P_1 , P_2 -> A_1
R_2 : P_1 , P_3 , P_4 -> A_1
R_3 : P_1 , P_5 , P_6 -> A_2
R_4 : A_1 , P_7 , P_8 -> A_3
R_5 : A_2 , A_3 , P_9 -> A_4

지식베이스에 위와 같은 지식이 있고 다음과 같이 차례로 규칙을 첨가해 본다.

① R_6 : P_1 , P_2 , P_7 -> A_5의 첨가 : R_6는 다음과 같이 변경되어 첨가되고 R_4도 변경된다.
R_6 : A_1, P_7 -> A_5
R_4 : A_5 , P_8 -> A_3

② R_7 : A_2 , P_5, P_9 -> A_6의 첨가 : R_7는 다음과 같이 변경되어 첨가되고 R_5도 변경된다.
R_7 : A_2 , P_9 -> A_6
R_5 : A_3 , A_6 -> A_4

③ R_8 : A_3 , P_{10} -> A_1의 첨가 : 순환이라 명시하며 첨가되지 않는다.

④ R_9 : P_{11} -> A_7의 첨가 : 수정없이 첨가된다.

⑤ R_{10} : ~P_{12} -> A_8의 첨가 : 수정없이 첨가된다.

⑥ R_{11} : A_8 , P_{12} , P_{13} -> A_9의 첨가 : 조건절에 P_{12}와 ~P_{12}가 있는 경우이므로 모순임을 알리며 첨가되지 않는다.

⑦ R_{12} : P_1 , P_5 , P_6 -> A_{10}의 첨가 : R_3와 R_{12}의 조건절이 같아서 A_2와 A_{10}의 관계를 점검한 후 모순이 발견되지 않으므로 지식 베이스내에서의 모순이 없음을 알리며 받아들일 것인지를 묻는다.

⑧ R_{13} : P_1 , ~P_8 -> A_1의 첨가 : P_8과 P_2의 관계가 모순관계는 아니지만, R_{13}을 삽입하면 첨가과정의 단계 6에서 R_4 : A_5 , P_8 -> A_3가 모순인 규칙으로 발견되므로 시스템은 지식의 첨가자에게 R_{13}을 첨가할 경우 R_4가 모순으로 변함을 알리고 R_{13}을 삽입할 것인지를 묻는다.

위 예의 각 과정에서 더 이상 연결되지 않는 규칙과 도달될 수 없는 규칙의 리스트를 보여준다. ⑧의 첨가가 끝난 후, R_5, R_9, R_{10}은 더 이상 연결되지 않는 규칙이고 진입 차수가 0인 R_1, R_2, R_3, R_9, R_{10}등은 입력 변수가 아니면 도달될 수 없는 규칙임을 알린다.

5. 알고리즘 분석

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

n : 지식베이스 내의 규칙의 수
l : 확실한 특성의 리스트의 수
m : 우발적 특성의 리스트의 수
k : 하나의 규칙 내에서 조건부의 최대 수
h : 하나의 리스트에서 항의 최대수

다음은 최악의 경우를 고려하여 분석해 본다. 즉 새로운 규칙이 모든 지식에 영향을 미친다고 가정하자. 이때 각 단계의 비교 횟수는 다음과 같다.

1단계 : 삽입하려는 규칙의 조건부의 프레미스들을 사실

로 놓고 전방 추론하여 실행부를 첨가한다. 이때 네트워크이므로 보통 적은 수의 규칙을 이용하여 추론하지만 최악의 경우 n개를 모두 참조한다고 가정하면 $O(n \cdot k)$.

2단계 : 의미적 모순을 검사하기 위해 조건부 내의 각 프레미스 끼리 비교하므로 k개의 조건부라면 $(k \cdot (k-1))/2$ 의 비교가 있다. 또한 순환을 검사하기 위하여 조건부의 각각의 프레미스와 실행부를 비교하므로 k번의 비교가 있다. $O(k^2 + k) \approx O(k^2)$.

3단계 : 삽입하려는 규칙의 조건부가 k개라면 이 k개에 대해 우발적 리스트를 참조하여 중복된 것을 제거하므로 $O(k \cdot h)$.

4단계 : 첨가할 규칙의 조건부와 각 규칙들의 조건부를 비교하므로 $O(n \cdot k)$.

5단계 : 첨가할 규칙과 실행부가 같은 규칙들을 비교하므로 $O(n)$. 실행부가 같은 규칙이 있는 경우에는 그 규칙의 조건부와 첨가할 규칙의 조건부가 모순 관계에 있는지를 우발적 리스트를 참조하여 점검하므로 $O(k \cdot h)$를 더해야 한다.

6단계 : 첨가된 규칙의 후속자들의 의미적 모순을 점검하는 과정은 보통 소수개의 규칙에 해당되나 최악의 경우 모든 규칙에 영향을 미친다고 가정해도 각 규칙의 우발적 리스트를 사용하여 규칙을 확장시킨 후 점검하므로 $O(n \cdot k \cdot h)$. 첨가할 규칙이 다른 규칙에 포함되는지를 점검하기 위해 각 규칙들과 첨가할 규칙의 조건부를 비교하므로 $O(n \cdot k)$.

7단계 : 첨가된 규칙의 실행부가 다른 규칙의 조건부에 있는지를 검사하며 동시에 진입 차수도 계산하므로 $O(n \cdot k)$.

8단계 : 도달될 수 없는 규칙은 규칙의 진입차수가 0인 경우이므로 $O(n)$이고, 더 이상 연결되지 않는 점은 후속자가 없는 경우이므로 $O(n)$.

따라서 5단계까지 각 단계를 모두 합한 결과는 $O(nk + k^2 + kh + nk + n + kh) = O(n(2k+1) + k(k+2h)) \approx O(nk+k^2+kh)$이며 k와 h는 보통 작은 상수이다. 그리고 새로운 규칙의 일관성을 점검한 뒤 이 규칙이 영향을 미치는 모든 규칙을 점검해 봐야한다. 결과적으로 이 알고리즘의 6단계까지의 모든 과정을 거친 후 시간 복잡도는 $O(nk + k^2 + kh + nkh) \approx O(nkh + k^2)$이라 나타낼 수 있으며 그 후에 독립적으로 검증되는 7단계까지 합해도 $O(nkh + k^2 + 2n)$ 이므로 $O(nkh + k^2)$이라 나타낼 수 있어 삽입과 제거 연산이 비교적 빠른 시간 내에 가능하다.

6. 결 론

이 논문은 지식의 일관성과 완결성을 유지하기 위한 원칙, 방법, 알고리즘을 제시하였다. 새로 첨가될 지식은 기존의 지식과의 중복, 모순, 순환, 도달되지 않는 점과 더 이상 연결되지 않는 점 등의 오류를 구조적 및 의미론적 측면에서 판별한 뒤 오류가 없는 경우 지식의 네트워크로 첨가된다. 그런데 이렇게 규칙이 첨가될 경우 네트워크의 구성, 링크의 생성과 제거가 자동적으로 행해지므로 전문가는 새로운 지식을 쉽게 첨가할 수 있다.

결국, 본 논문에서는 전문가 시스템의 추론의 신뢰도를 높이고 효율성을 증가시키기 위하여 오류 제거의 편집 과정을 통해 지식의 일관성 및 완결성을 유지하고, 네트워크로 구조화하여 추론 엔진이 실행도중 실행해야 할 규칙을 선정하는데 있어서 모든 규칙을 검증하지 않고 관련된 규칙만을 탐색하도록 하여 효율성을 증가시켰고, 제안된 시스템이 맨-머쉰 인터페이스의 효과를 나타냄으로써 전문가가 지식을 쉽게 첨가할 수 있도록 하였다.

참 고 문 헌

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

[2] 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.

[3] 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.

[4] 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.

[5] Gerard Ellis, "Compiling Conceptual Graphs", *IEEE trans. on Knowledge and data engineering*, vol 7, no 1, pp. 68-81, feb, 1995.

[6] P. LeBeux, D. Fontaine, "Un systeme d'acquisition des connaissances pour systemes experts", *Technique et Science Informatique*, vol 5, no.1, pp7-20, 1986.

[7] 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.

[8] 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.

[9] D.L.Nazareth, "Issues in the Verification of Knowledge in Rule-Based Systems", *I.J.Man-Machine Studies*, vol. 30, pp. 255-271, 1989.

[10] D.L.Nazareth and M.H.Kennedy, "Verification of Rule-Based Knowledge Using Directed Graphs", *Knowledge Acquisition*, vol.3, pp. 339-360, 1991.

[11] 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.

[12] T.A. Nguyen, N.A.Perkins, T.J.Laffey and D.Pecora,"Checking an Expert system knowledge base for consistency and completeness", *Proc. 9th IJCAI*, LA, Califonia, pp.375-378, 1985.

[13] 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.

[14] 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.

[15] G.Valiente, "Verification of knowledge based redundancy and subsumption using graph transformations", *Int'l J. Expert System*, vol.6, no.3, pp341-355, 1993.

서 의 현

1980년 이화여자대학교 수학과 (이학사)

1980년~1982년 한국개발연구원, 연구원

1985년 프랑스 꽁삐엔느 대학교 컴퓨터공학과 (공학석사)

1988년 프랑스 꽁삐엔느 대학교 컴퓨터공학과 (공학박사)

1990년~현재 목원대학교 컴퓨터공학과 부교수

관심분야 : 인공지능, 지식 공학, HCI