지식 획득 시스템을 갖춘 전문가 시스템의 구현

서 의 현†

요 약

전문가 시스템은 특정 영역의 지식을 기반으로 추론을 행한다. 이 때 지식의 정확성 및 일관성 여부는 추론 결과의 신뢰도와 직결된다. 따라서 전문가 시스템에는 다양한 지식을 호출하고 지식의 일관성을 유지시켜줄 수 있는 기능들이 필요하다. 본 논문에서는 지식베이스는 물론 데이터 베이스 내의 선언적 지식과 절차적 지식들을 호출할 수 있는 기능을 갖춘 전문가 시스템이 구축되었다. 이와 함께 지식이 지식 베이스에 첨가될 때 기존 지식과의 모순, 중복, 순환, 도달될 수 없는 규칙, 더 이상 연결되지 않는 규칙 등을 조사한 후 일관성이 유지될 때만 지식이 첨가되는 지식 획득 시스템이 구현되었다. 결과적으로 전문가 시스템은 다양한 지식을 호출할 수 있고, 추론 결과의 신뢰도가 높아졌으며 지식 획득 시스템은 인간과 컴퓨터의 인터페이스 기능을 강화함으로써 사용자가 지식을 지식베이스에 쉽게 첨가할 수 있도록 하였다.

An Implementation of Expert System with Knowledge Acquisition System

Euy-Hyun Suh†

ABSTRACT

An expert system executes the inference, based on the knowledge of specific domain. The reliability on the results of inference depends upon both the consistency and accuracy of knowledge. This is the reason why expert system requires the facilities which can practice an access to the various kinds of knowledge and maintain the consistency and accuracy of knowledge. This paper is to implement an expert system permitting an access of declarative and procedural knowledge in the knowledge base and in the data base. This paper is also to implement a knowledge acquisition system which adds the knowledge only if its accuracy and consistency are maintained, after verifying the potential errors such as contradiction, redundancy, circulation, non-reachable rule and non-linked rule. In consequence, the expert system realizes a good access to the various sorts of knowledge and increases the reliability on the results of inference. The knowledge acquisition system contributes to strengthening man-machine interface that enables users to add the knowledge easily to the knowledge base.

1. 서 론

전문가 시스템은 기본적으로 추론엔진 및 제어 모듈, 지식베이스와 사용자 인터페이스로 구성된다. 이 시스템은 규칙과 사실들의 매칭에 의한 일련의 규칙을 실행하여 추론을 수행하는 과정에서 유용한 형태의 지식들을 생성하여 사용자에게 제공한다. 이렇게 생성된 지식이 유용하기 위해서는 지식 공학자가 해당 분야의 많은 전문지식을 모아서 원하는 형태의 추론을 제공할 수 있는 지식 베이스가 구축되어야 한다. 또한 지식의 효율적인 추론을 위하여 좋은 추론 메카니즘과 전문가 시스템을 편리하게 이용하기 위한 인터페이스가 제공되어야 한다.

이러한 맥락에서 전문가 시스템을 쉽게 구축할 수 있는 전문가 시스템 쉘(shell)이 제공되고 있다. 많이 사용되는 쉘에는 OPS5, KES, EXPERT, CLIPS[6]등이

† 정 회 원 : 목원대학교 컴퓨터공학과 교수
논문접수 : 1999년 10월 20일, 심사완료 : 2000년 5월 6일

있다. 이러한 쉘들은 추론 메카니즘을 제공하며 지식베이스를 쉽게 구축하고 유지할 수 있는 개발 환경을 제공하고 있어 지식 공학자의 노력을 많이 절감시켜준다. 그러나 이러한 쉘에서 전문가 시스템의 신뢰도의 성능을 좌우하는 지식베이스 내의 지식의 검증은 극히 기본적인 것에 불과하며 지식의 일관성 및 정확성 여부는 전적으로 지식의 제공자에 의하여 결정된다.

그런데 지식의 양은 방대한 경우가 많기 때문에 지식 제공자(전문가)나 지식 공학자가 내용 전체를 파악하기가 어렵고, 전문가는 때때로 직관적으로 생각하므로 추론에서의 많은 과정을 생략한 채 지식을 지식베이스에 삽입할 수 있다. 또한 지식베이스는 점진적인 방식으로 오랜 기간에 걸쳐 구축되고 여러 명의 전문가가 지식을 함께 저장하기 때문에 지식의 중복, 순환과 모순의 문제가 야기될 수 있다. 이러한 문제들을 해결하기 위해서는 전문가 시스템이 지식베이스를 사용하기 전에 정확성을 검증하여 발견된 오류를 자체 수정하거나 자체 수정이 불가능할 경우 전문가에게 문의하여 수정하는 시스템이 요구된다.

이러한 지식베이스의 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[4]. 그런데 이 방식은 부분적인 일관성만을 자동으로 처리할 수 있을 뿐이다.

의존 관계표를 사용하여 일관성과 완결성을 점검하는 방법[15]은 열거 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 규칙들은 추론 과정 중 연결된 규칙이 많아 지수 함수적인 성능을 나타낸다[12].

쌍 검증(Pairwise Checking) 방식에서는 생성된 절들이 적용 가능한 규칙들과 비교된다[9, 11]. 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[10].

그래프 표현 방식은 프레미스의 개념적인 의존 관계를 나타내기 쉬운 표현법으로서 방향 그래프(Directed Graph)[13], 추론 그래프(Inference Graph)[16], 하이퍼 그래프(Hypergraph)[21]등의 방법이 있다. 그러나 이러한 방법들은 복합절과 단순절 사이의 의존 관계를 나타내기 어렵다[18]. 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)[18]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 구조로서 지식을 체계적으로 분류하여 계층적으로 저장하고 편집 방법을 통해 중복을 제거한다[3, 5]. 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[10, 14]은 일관성 검진이 정해진 실행 순서로 진행되어야 하는데 형식이 완전하지 못하면 제약 조건이 적용되지 못하고 수행시간도 많이 걸리는 단점이 있다.

신뢰 유지 시스템(Belief Maintenance System)[7, 8, 17, 19, 20]은 추론도중 새로운 지식을 획득했을 경우 기존 지식들과의 모순을 찾아내어 진행중인 지식들 사이의 일관성을 유지하는 시스템이다. 이 시스템은 비단조 추론 혹은 불완전한 지식을 가지고 추론하는 경우에 추론 엔진과 더불어 사용되고, 실행 중에 있는 지식들의 일관성을 동적으로 유지시키는 시스템이므로 영구적이고 정적인 지식베이스 내용을 수정하는 것은 많은 주의가 요구되며 수행 시간도 많이 걸리는 단점이 있다.

따라서 본 논문은 새로운 지식이 기존의 지식베이스에 첨가되기 전에 새로운 지식의 정확성 및 일관성을 검증한 후 일관성이 유지될 경우에만 지식을 지식베이스에 첨가하는 지식 획득 시스템을 구축하였다. 이와 함께 지식베이스 및 데이터 베이스의 지식을 호출할 수 있는 전문가 시스템을 구축하여 지식 획득 시스템과 연계시켰다. 지식 획득 시스템에서는 확실한 특성의 리스트와 우발적 특성의 리스트를 생성하고 검증 방법 및 단계를 개선하는 형태로 쌍 검증 방식을 보완한다.

이 리스트들과 개선된 검증 방법은 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. 본 논문은 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 획득 시스템을 제시, 구축하였다. 이와 더불어 지식 베이스와 데이터 베이스의 다양한 지식을 호출하여 추론하는 추론 엔진을 구축하였다. 모든 시스템은 C++언어와 Oracle SQL을 사용하여 Sun Sparc 5.5에 구축되었다.

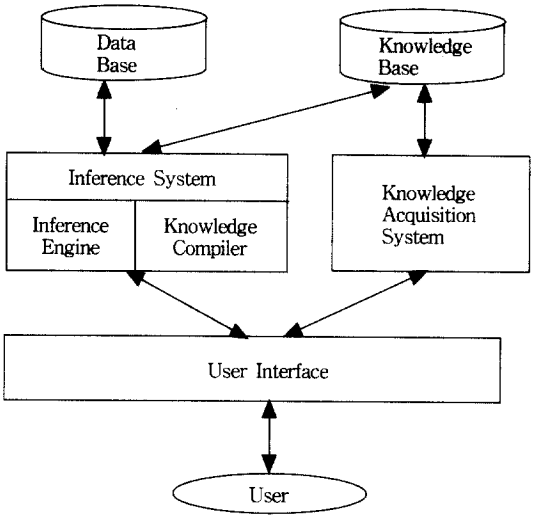
지식 및 데이터 베이스와 사용자 인터페이스 등 시스템의 구성요소를 2장에서 설명한 후 3장에서는 지식 베이스의 구축에 대해 소개한다. 4장과 5장에서는 추론 시스템 그리고 지식획득 시스템에 대해서 자세히 설명하고 구현된 시스템을 실행한 예를 6장에서 보인다. 마지막으로 결론에서는 지식 획득 시스템을 갖춘 전문가 시스템을 구축함으로써 확인된 결과를 요약하고 향후 과제를 다룬다.

2. 시스템 개요

시스템은 (그림 1)과 같이 구성된다.

2.1 지식베이스

지식베이스의 기본 구조는 생성 규칙이다. 생성규칙은 규칙들을 모듈화하여 나타낼 수 있고, 비절차적 방식으로도 규칙들로서 저장된 지식을 사용할 수 있을 뿐 아니라 여러 종류의 지식을 쉽게 표현할 수 있기 때문이다. 지식은 조건 부분과 실행 부분으로 표현되며 일반 지식과 절차적 지식도 같은 방식으로 나타낼 수 있도록 하였다. 이러한 생성 규칙들의 실행 효율을 증가시키기 위하여 내부 표현 구조는 네트워크 구조로 표현하였다.



(그림 1) 시스템 구조

2.2 데이터 베이스

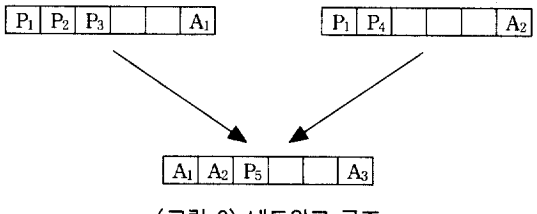
관계형 데이터 베이스로서 Oracle 7.3.3을 사용하여 일반적인 정보들을 표현하였다. 전문가 시스템은 필요한 경우 데이터 베이스의 정보들을 호출할 수 있도록 하였다. 예를 들면 병원의 데이터 베이스에는 환자의 병력에 대한 정보들이 저장되어 있으므로 추론 도중 환자의 진료를 위해 이들을 호출할 수 있어야 한다.

2.3 추론 시스템

추론 시스템은 추론 엔진과 지식 컴파일러로 구성된다.

생성 규칙 기반 시스템에서 많이 사용하는 추론 엔진의 기법은 전방 추론(forward chaining)과 후방 추론(backward chaining)이다[22]. 본 시스템에서는 확신도에 의해 추론의 순서가 결정되는 전방 및 후방 추론을 구축하였다. 추론 방식은 사용자가 선택할 수 있다.

지식 컴파일러는 전문가 시스템이 실행되기 전에 지식을 네트워크의 내부 표현 방식으로 컴파일한다. 지식의 네트워크 표현 방식은 (그림 2)와 같다.



(그림 2) 네트워크 구조

네트워크 표현 방식은 하나의 규칙 내부에 이 규칙과 연결되어지는 규칙의 정보를 포함하고 있다. 이 방식은 추론이 실행될 때 이러한 정보를 이용함으로써 모든 지식을 검색할 필요없이 꼭 필요한 지식만 검색해 봄으로써 추론의 속도를 증가시킬 수 있다.

2.4 지식 획득 시스템

지식 획득 시스템은 새로운 지식을 지식베이스에 추가하고자 할 경우, 지식이 추가되기 전에 정확성을 검증한다. 이는 만약 오류가 존재한다면 오류를 자체 수정하거나 자체 수정이 불가능할 경우 전문가에게 문의하여 오류를 수정하도록 한 후 지식이 첨가되도록 하는 시스템이다. 이 시스템의 목표는 지식의 정확성 및 일관성을 유지시켜 전문가 시스템의 신뢰도를 높이는 것이다.

2.5 사용자 인터페이스

사용자가 전문가 시스템을 쉽게 이용할 수 있도록 대화형의 인터페이스가 구축되었다. 전문가 시스템이 사용자에게 질문을 하면 사용자는 주어진 답안 중에서 하나를 선택하거나 간단한 스트링이나 숫자를 입력하든지 혹은 예(true), 아니오(false) 형태로 대답하면 된다.

3. 지식 베이스

지식 베이스는 다양한 종류의 지식을 표현할 수 있는 생성 규칙들로 구성된다. 본 시스템에서는 선언적인 지식과 절차적인 지식이 규칙 내에 표현될 수 있도록 하였다. 또한 절차적 지식에는 데이터 베이스내의 지식을 호출, 삽입, 제거 및 수정할 수 있는 기능도 포함되어 있다.

3.1 규칙의 표현

규칙의 표현은 (그림 3)과 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> ::=<string_type> <question_type> {{<letter> | <digit>}1^255
<string_type> ::= i | n | f
            [참고] i : initial
                   f : goal
                   n : non-initial
<ques_type> ::= 1 | 2 | 3
            [참고] 1 : .....isn't it  ?
                   2 : What is..... ?
                   3 : .....doesn't it ?
<letter> ::= A| ... |Z a|...|z
<digit> ::= 0|1|...|9
<operator> ::= '=' | '!=' | '<' | '<=' | '>' | '>=' | <indicator_function_result>
<indicator_function_result> ::= n | y
            [참고] n : 결과 없음.
                   y : 결과 존재.
<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 3) 규칙의 표현

3.2 절차 부가(Procedural Attachment)

생성 규칙 표현법의 장점들 중 하나는 다양한 지식을 표현할 수 있다는 것이다. 전문가 시스템은 필요한 경우 함수를 호출하여 비결정적인 값을 계산할 수 있어야 한다.

예를 들면,

```
rule1 (0.9) <modeler>
  if    type of model, s=triangle
        surface, x=calcul_surface_tri(base, height)
        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 해당 함수를 진행시킨다.

3.3 데이터 베이스의 지식에 대한 호출

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

따라서 본 시스템은 데이터 베이스 내의 정보의 호출, 추가, 제거 및 수정할 수 있는 기능을 갖추었다. 관계(relation), 엔티티(entity), 키(key), 레코드(record)와 호출된 값을 보관할 수 있는 변수의 이름들을 가지고 조작할 수 있다.

예를 들면,

```
rule2 (0.8) <modeler>
 if
      eq1, i y a(relation_name, equation, key_col:edge1_id, return_value)
      eq2, i y a(relation_name, equation, key_col:edge2_id, return_value)
      calcul_state, x = calcul_parall(eq1, eq2)
      calcul_state, s = intersection
 then delete_edge, i n s(relation_name, key_col, edge2_id)
```

이것은 절차 부가의 일종이므로 수행과정은 절차 부가와 동일하다.

4. 추론 시스템

추론 시스템은 추론 엔진과 지식 컴파일러로 구성되어 있다.

4.1 추론 엔진

4.1.1 전방 추론

전방 추론은 [알고리즘 1]에 기술되었다.

```
function forward_chaining (a_fact)
begin
  if (a_fact ∈ facts) 끝(성공)
  if (forward_chaining_cycle(rules, a_fact) = true) 끝(성공).
                                          else 끝(실패).
  end

function forward_chaining_cycle(rules, a_fact)
begin
  if (IsEmpty(rules)) return(false)
  a_rule <- 가능한 rules 중에서 priority가 가장 높은 규칙
  while (a_rule = not null)
    begin
    if (a_rule의 모든 조건부 ∈ facts)
       begin
        if (a_rule의 실행부 ∉ facts) facts <- facts + a_fact의 실행부
        if (a_rule의 실행부 = a_fact) return(true)
        base_rules <- rules - a_rule
        if (forward_chaining_cycle(base_rules, a_fact)=true) return(true)
                                                  else return(false)
       end
   a_rule <- 가능한 rules 중에서 priority가 가장 높은 규칙
   end
end
```

[알고리즘 1] 전방 추론

4.1.2 후방 추론

후방 추론은 [알고리즘 2]에 기술되었다.

```
function backward_chaining(goal)
begin
   if (verify_fact (goal)=true)  끝(성공)
                          else 끝(실패)
end

boolean function verify_fact(a_fact)
begin
 if (a_fact ∈ facts) return(true)
 a_rule <- 가능한 rules 중에서 priority가 가장 높은 규칙
 while (a_rule = not null )
       begin
       if (a_rule의 실행부 = a_fact)
              if (verify_cond(a_rule)=true)
                     begin
                     facts <- facts + a_rule의 실행부
                     if (a_fact = a_rule의 실행부) return(true)
                     end
       a_rule <- 가능한 rules 중에서 priority가 가장 높은 규칙
       end
 return(false)
end

boolean function verify_cond(a_rule)
begin
 fact <- a_rule의 첫 번째 조건
 fact_no <- 0
 while (fact = not null)
       begin
       fact_no <- fact_no +1
       true_tab[fact_no] <- false
       if (fact ∈ facts) true_tab[fact_no] <- true
              else if (verify_fact(fact) = true) true_tab[fact_no]=true
                                          else return(false)
       fact <- a_rule 의 다음 조건
       end
  if (true_tab[1]부터 true_tab[fact_no]의 모든 원소 = true) return(true)
                                                  else return(false)
end
```

[알고리즘 2] 후방 추론

4.2 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 있는 장점을 가지기 때문이다. 지식은 오류를 점검하는 편집 과정을 거쳐 오류가 없으면 네트워크 구조에 추가되며 최종적으로 지식베이스에 저장된다.

5. 지식 획득 시스템

전문가 시스템은 규칙과 사실들을 매칭시켜 일련의 규칙을 실행시키는 방식으로 추론을 수행하는데 이 때 추론된 지식의 신뢰도를 높이기 위해서는 지식의 정확성 및 일관성이 유지되어야 한다. 따라서 이 시스템에서는 확실한 특성의 리스트와 우발적 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.

5.1 확실한 특성의 리스트와 우발적 특성의 리스트

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

- **정의 1** : 네트워크의 규칙을 R_i, i = 1..n으로 표시하고, 하나의 규칙이 조건 부분(P)과 실행 부분(A)으로 표현되면 규칙 R_i의 조건부분의 모든 프레미스는 Set R_i라 정의한다.

- **정의 2** : $R_1 -> R_2 -> ... -> R_n$이고 R_n의 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 특성(Certain(E))이라 정의한다. 표현방법은 E -> P이다. 다음 3조건 중 하나를 만족하면 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.

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

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

  $\text{Certain}(P_{ji}) = 0$ 만약 P_{ji}에 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** : 우발적 특성은 일어날 수도 있고 그렇지 않을 수도 있는 특성을 말한다. 만약 P의 실행이 E의 실행에 기여한다면, P는 E의 우발적 특성(Eventual(E))이라 정의한다. 만약 아래의 조건 중 하나를 만족하면 P는 E의 우발적 특성이다.

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

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

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

  $\text{Eventual}(P_{ji}) = 0$: 만약 Pji에 관계된 경로가 없다면.

  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.

결국 다음과 같은 결과가 얻어진다. Certain(E) ⊂ Eventual(E).

5.2 지식베이스의 오류

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

5.2.1 중 복

지식의 중복은 다음과 같이 점검된다.

- 구조적 중복과 수정
  - Set R_1 = Set R_2 and $A_1 = A_2$: 하나의 규칙은 제거한다. 단, A는 실행부를 나타낸다.
  - Set R_1 ⊂ Set R_2 and $A_1 = A_2$: R_1이나 R_2 둘 중 하나의 규칙만 선택하는데 R_2를 선택하는 것이 바람직하다.
  - $A_1 \in$ Set R_2 and #(Set $R_1 \cap$ Set R_2) >= 1 : Set R_2 = Set R_2 - Set R_1

- 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_2이거나 Set R_1 ⊂ Set R_2이고 A_1과 A_2중 어느 하나가 다른 하나의 확실한 특성 또는

우발적 특성의 관계에 있을 때, 확실한 특성 또는 우발적 특성을 실행부로 가진 규칙을 제거한다.

5.2.2 모 순

- 구조적 모순과 수정
  - Set R_1 = Set R_2 and $A_1 \neq A_2$: 만약 A_1과 A_2가 논리적 모순 관계이면 규칙 중 하나를 버려야 한다. 이 경우 지식 제공자에게 수정하라는 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 특성을 말한다.
  - Set $R_1 \subset$ Set R_2 and $A_1 \neq A_2$: 만약 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}.
- 의미적 모순과 수정

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

5.2.3 순 환

하나의 규칙 내에 프레미스 P를 첨가하려는 데 실행부 A가 P의 우발적 특성 중 하나인 경우, 시스템은 사용자에게 경고하고 프레미스를 첨가하지 않고 규칙의 첨가자가 선택하도록 한다.

5.2.4 더 이상 연결되지 않는 규칙

한 규칙의 실행부가 목표가 아니고, 이 규칙의 실행부가 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.

5.2.5 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.

5.3 새로운 지식의 첨가 및 제거

5.3.1 지식을 첨가하기 전의 오류점검

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

- 단계 1 : 지식베이스의 규칙들 R_i와 첨가하려는 규칙 R을 비교한다. 모든 R_i에 대해 만약 Set $R_i \subset$ Set R, $A_i \notin$ Set R이면 Set R = Set R $\cup$ {A_i}이다.
- 단계 2 : 첨가하려는 규칙 내부에서 조건 부분들 사이에 의미적 모순 또는 순환을 검증하고 제거한다. 조건부의 프레미스들을 서로 비교하여 모순 관계를 점검하고 만약 모순이 발견되면 전문가에게 수정을 요구한다. 조건부와 실행부에 같은 프레미스가 있으면 순환이다.
- 단계 3 : 불가피한 프레미스만 남겨두고 Set R을 감축시킨다. 즉 지식의 구조적 및 의미적 중복을 제거하는 것이다. Set R′ = Set R - $\cup$ {우발적 프레미스 (P_i)}
- 단계 4 : 첨가하려는 R은 조건 부분들이 같은 규칙들과 비교된다. 즉 구조적 중복과 모순을 제거하는 것이다. 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 것인지를 검사한다.
- 단계 5 : 규칙 R은 실행 부분이 같은 규칙들과 비교된다. 즉 구조적 중복과 모순을 제거하는 것이다. 만약 조건 부분이 다른 R_i의 Set(R_i) 중 Set(R)과 모순되는 것이 있으면 모순을 지적하고 전문가에게 수정을 요구한다.

5.3.2 규칙의 첨가와 네트워크의 수정

규칙은 위의 5단계를 거쳐 오류를 제거하고 수정되어 지식베이스에 첨가될 수 있다. 그러나 이 새로운

규칙이 첨가됨으로써 지식베이스의 기존의 규칙들을 더욱 간소화시킬 수 있는지를 먼저 점검한 후 새로운 규칙이 첨가되도록 한다.

- 단계 6 : 새로 삽입될 규칙의 실행부가 다른 규칙의 조건부라면 새로운 규칙을 삽입함으로써 기존의 규칙들이 모순을 일으키지 않는지를 점검한다. 즉 구조적 및 의미적 모순을 제거하는 것이다. 이 때 모순이 발견되면 전문가에게 알린다. 또한 첨가할 규칙 R의 $Set(R) \subset Set(R_i)$인 R_i의 조건부를 $Set\ R_i = (Set\ R_i - Set\ R) \cup \{A\}$로 수정하여 구조적 중복을 제거한다. 그리고 진입 차수와 진출 차수도 수정한다.
- 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.

새로운 규칙은 위의 8단계를 모두 거친 후 지식 베이스에 첨가된다.

5.3.3 규칙의 제거

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.

5.3.4 예

구축된 지식 획득 시스템에 의해 다음과 같은 규칙을 첨가시키고 실행시켜 나타난 시스템의 처리 결과는 다음과 같다.

$R_1 : P_1, P_2 \rightarrow A_1$
$R_2 : P_1, P_3, P_4 \rightarrow A_2$
$R_3 : P_1, P_5, P_6 \rightarrow A_3$
$R_4 : A_1, P_7, P_8 \rightarrow A_4$
$R_5 : A_3, A_4, P_9 \rightarrow A_5$

지식베이스에 위와 같은 지식이 있고 다음과 같이 차례로 규칙을 첨가해본다.

① $R_6 : P_1, P_2, P_7 \rightarrow A_6$의 첨가 : R_6는 다음과 같이 변경되어 첨가되고 R_4도 변경된다.

$R_6 : A_1, P_7 \rightarrow A_6$
$R_4 : A_6, P_8 \rightarrow A_4$

② $R_7 : A_3, P_5, P_9 \rightarrow A_7$의 첨가 : R_7는 다음과 같이 변경되어 첨가되고 R_5도 변경된다.

$R_7 : A_3, P_9 \rightarrow A_7$
$R_5 : A_4, A_7 \rightarrow A_5$

③ $R_8 : A_4, P_{10} \rightarrow A_1$의 첨가 : 이 경우 첨가 단계 1에 의해 A_4는 A_1, P_7, P_8으로 대치되므로 R_8은 $A_1, P_7, P_8\ P_{10} \rightarrow A_1$ 되나 단계 2에서 조건부와 실행부에 A_1이 같이 존재하므로 순환이라 명시하며 첨가되지 않는다.

④ $R_9 : P_{11} \rightarrow A_8$의 첨가 : 수정없이 첨가된다.

⑤ $R_{10} : \neg P_{12} \rightarrow A_9$의 첨가 : 수정없이 첨가된다.

⑥ $R_{11} : A_9, P_{12}, P_{13} \rightarrow A_{10}$의 첨가 : R_{10}에 의해, R_{11}은 $\neg P_{12}, P_{12}, P_{13} \rightarrow A_{10}$이 된다.
이 때 조건절에 P_{12}와 $\neg P_{12}$가 있는 경우이므로 모순임을 알리며 첨가되지 않는다.

⑦ $R_{12} : P_1, P_5, P_6 \rightarrow A_{11}$의 첨가 : R_3와 R_{12}의 조건절이 같아서 A_3과 A_{11}의 관계를 점검한 후 모순이 발견되지 않으므로 지식 베이스 내에서의 모순이 없음을 알리며 받아들일 것인지를 묻는다. 받아들이기를 원하면 $A_3 \rightarrow A_{11}$로 저장된다.

⑧ $R_{13} : P_1, \neg P_8 \rightarrow A_1$의 첨가 : P_8과 P_2의 관계가 모순관계는 아니지만, R_{13}을 삽입하면 첨가과정의 단계 6에서 $R_4 : A_6, P_8 \rightarrow A_4$가 모순인 규칙으로 발견되므로 시스템은 지식의 제공자에게 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_2, 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.

5.4 성능 분석

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

n : 지식베이스내의 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 사실로 놓고 전방 추론하여 실행부를 첨가한다. 이때 네트워크이므로 보통 적은 수의 규칙을 이용하여 추론하지만 최악의 경우 n개를 모두 참조한다고 가정하면 비교 횟수는 $O(n \cdot k)$이다.
- 2단계 : 의미적 모순을 검사하기 위해 조건부 내의 각 프레미스끼리 비교하므로 k개의 조건부라면 $(k \cdot (k-1))/2$의 비교가 있다. 또한 순환을 검사하기 위하여 조건부의 각각의 프레미스와 실행부를 비교하므로 k번의 비교가 있다. 따라서 비교 횟수는 $O(k^2 + k) \approx O(k^2)$이다.
- 3단계 : 삽입하려는 규칙의 조건부가 k개라면 이 k개에 대해 우발적 리스트를 참조하여 중복된 것을 제거하므로 비교 횟수는 $O(k \cdot h)$이다.
- 4단계 : 첨가할 규칙의 조건부와 각 규칙들의 조건부를 비교하므로 횟수는 $O(n \cdot k)$이다.
- 5단계 : 첨가할 규칙과 실행부가 같은 규칙들을 비교하므로 횟수는 $O(n)$이다. 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 : 첨가될 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 횟수는 $O(n \cdot k)$이다.
- 7단계 : 첨가될 규칙의 실행부가 다른 규칙의 조건부에 있는지를 검사하며 동시에 진입 차수도 계산하므로 비교횟수는 $O(n \cdot k)$이다.
- 8단계 : 도달될 수 없는 규칙은 규칙의 진입차수가 0인 경우이므로 비교횟수는 $O(n)$이고, 더 이상 연결되지 않는 절은 후속자가 없는 경우이므로 $O(n)$이다.

따라서 5단계까지 각 단계를 모두 합한 비교 횟수는 $O(nk + k^2 + kh + nk + n + kh) = O(n(2k + 1) + k(k + 2h)) \approx O(nk + k^2 + kh)$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. 구현 및 예제

시스템은 C++언어와 Oracle SQL을 사용하여 구축되었다. 그리고 15개의 프레미스와 8개의 규칙으로 구성된 지식을 사용하여 테스트해 보았다.

6.1 프레미스

다음은 스트링의 집합이다.

```
11
1i1 This is a flower.
2n2 the color(red, blue, green).
3n2 the number of petal.
4n1 The number of petal is known.
5n2 the result of calculation.
6i3 It has roots.
7f1 This is a rose.
8f1 This is a lily_of_the_valley.
9n2 the name of flower.
10n1 The calculation is known.
11n1 The color is known.
```

다음은 지식 베이스에서 프레미스의 내용이다. 모두 15개의 프레미스로 되어 있다.

```
15
 1  1 t
 2  2 s = red
 3  4 s = known
 4  5 x y calcule_f
 5  3 n > 0
 6  5 n > 0
 7  6 t
 8  7 t
 9  2 s = blue
10  8 x n print(lily_of_the_valley)
11  9 x y demand_name_of_type
12  2 i  y r(flower, 1 a_flower, 1 a_name : s : v : t9, a_color)
13 10 t
14 11 t
15  8 t
```

6.2 규 칙

다음은 전문가에 의해 제공된 지식이다.

```
rule1 (1.0) kl if 1t 14t  2t  3t  then 4t
rule2 (1.0) kl if 5t  then 3t
rule3 (1.0) kl if 13t  6t  7t  then 8t
rule4 (1.0) kl if 4t then 13t
rule5 (1.0) kl if 1t then 11t
rule6 (1.0) kl if 11t then 12t
rule7 (1.0) kl if 12t then 14t
rule8 (1.0) kl if  1t 14t  9t  3t then 10t
```

위의 지식을 하나씩 지식베이스에 첨가하면 지식 획득 시스템에 의해 kl이라는 지식베이스에 다음과 같이 첨가된다.

```
1 (1.0) 3 14t  2t  3t  4t
2 (1.0) 1  5t  3t
3 (1.0) 3 13t  6t  7t  8t
4 (1.0) 1  4t 13t
5 (1.0) 1  1t 11t
6 (1.0) 1 11t 12t
7 (1.0) 1 12t 14t
8 (1.0) 3 14t  9t  3t 10t
```

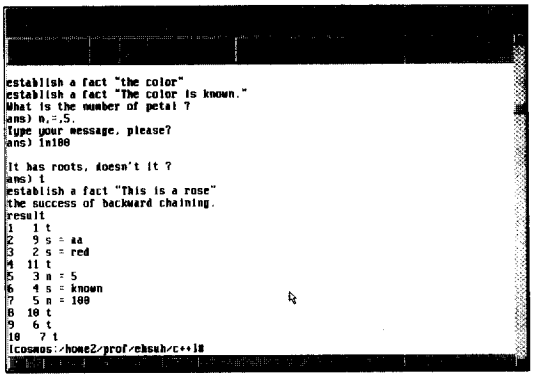
다음은 컴파일되어 네트워크로 표현된 지식이다.

```
1  1.0 3 14t  2t  3t  0t  0t  0t  0t  4t  4  0  0  0  0
2  1.0 1  5t  0t  0t  0t  0t  0t  0t  3t  1  8  0  0  0
3  1.0 3 13t  6t  7t  0t  0t  0t  0t  8t  0  0  0  0  0
4  1.0 1  4t  0t  0t  0t  0t  0t  0t 13t  3  0  0  0  0
5  1.0 1  1t  0t  0t  0t  0t  0t  0t 11t  6  0  0  0  0
6  1.0 1 11t  0t  0t  0t  0t  0t  0t 12t  7  0  0  0  0
7  1.0 1 12t  0t  0t  0t  0t  0t  0t 14t  1  8  0  0  0
8  1.0 3 14t  9t  3t  0t  0t  0t  0t 10t  0  0  0  0  0
```

6.3 실 행

다음은 실행 예이다. ans)는 사용자가 대답한 부분이다.

```
Select your goal, please ? : 7 This is a rose.
                             8 This is a lily_of_the_valley.
ans) 7t
Select an inference engine, please? : forward chaining : 1
                                      backward chaining : 2
ans) 2
This is a flower, isn't it ?
ans) t
Answer the demand_name_of_type
ans) 1saa
                ; 데이터 베이스에서 aa 라는 type의 꽃 색깔을 알아옴.
establish a fact "the color."
establish a fact "The color is known."
What is the number of petal?
ans) n,=,5.
Type your message, please?
ans) 1n100
                ; clacul_f 함수가 실행됨.
It has roots, doesn't it?
ans) t
establish a fact "This is a rose"
the success of backward chaining.
result
1   1 t
2   9 s = aa
3   2 s = red
4  11 t
5   3 n = 5
6   4 s = known
7   5 n = 100
8  10 t
9   6 t
10  7 t
```



7. 결 론

본 논문은 다양한 분야에 적용할 수 있는 전문가 시스템을 제안하고 구축하였다. 이 전문가 시스템은 지식베이스 내의 특정 영역의 지식과 데이터 베이스의 선언적 지식 및 절차적 지식을 호출하여 계산한 정보 등을 바탕으로 새로운 지식을 추론한다. 이러한 추론 결과의 신뢰도를 높이기 위해서 지식의 일관성 검증을 위한 지식 획득 시스템이 첨가되었다. 지식 획득 시스템에서는 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. 이와같이 새로운 규칙이 첨가될 경우, 오류 점검이 지식 획득 시스템에 의해 자동적으로 행해지므로 지식 제공자는 새로운 지식을 일관성을 유지시키며 쉽고 간단히

첨가할 수 있다.

결국 본 논문에서는 전문가 시스템 추론 결과의 신뢰도가 전적으로 지식 제공자에 의해 결정되는 방식을 피하고 지식 획득 시스템 내에서 지식의 오류를 자동적으로 찾아내어 지적하고 수정하게 함으로써 추론 결과의 신뢰도를 제고하였다. 또한 지식 제공자가 지식을 제공할 때 모든 지식을 점검하지 않으면서도 일관성 있게 지식을 첨가할 수 있으므로 결국 인간과 컴퓨터사이의 인터페이스의 기능도 향상되었다고 할 수 있다. 그리고 다양한 종류의 지식이 지식 컴파일러에 의해 추론이 실행되기 전에 네트워크 구조로 컴파일되어 꼭 필요한 지식만을 탐색함으로써 전문가 시스템의 추론의 효율성이 향상되었다. 본 시스템은 몇 개의 실험 데이터만을 사용하여 테스트해 보았다. 따라서 향후 연구 과제는 특정 분야에 대한 지식베이스를 구축하고 이 논문에서 제시한 지식획득 시스템과 전문가 시스템에 적용해 보는 것이다. 이와 같은 본 논문의 실용화 시도는 논문가치를 제고시키기 위하여 필수적이라는 점에서 더욱 그렇다.

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] 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, pp.4-13, 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] Giarratano, Riley, "Expert Systems : Principles and Programming," PWS Publishing Company ed., 1997.

[7] R. Guo, Hou Zhengfeng, "Issues of circulations in nonmonotonic reason maintenance systems," *IEEE international Conference on systems, Man and Cybernetics*, Vol.2, pp.1430-1435, 1996.

[8] Y.Kudo, T.Murai, T.Da-Te, "The correspondence of belief changes in logical settings and the possibilistic framework," *Proccedings KES'98. Second International Conference on Knowledge-Based Intelligent Electronic Systems*, 21-23 April 1998, Vol. 2, pp.221-229, 1998.

[9] P. LeBeux, D. Fontaine, "Un systeme d'acquisition des connaissances pour systemes experts," *Technique et Science Informatique*, Vol.5, No.1, pp.7-20, 1986.

[10] 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

[11] 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.

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

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

[14] 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.

[15] 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.

[16] 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.

[17] V.Poznanski, "Dempster-Shafer ranges for an RMS," *IEE Colloquium on Reasoning Under Uncertainty*, pp.5/1-5/2, 1990.

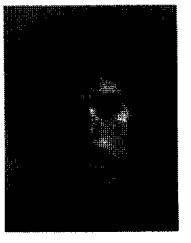
[18] M. Ramaswamy, S. Sarkar, Ye-Sho Chen, "Using directed hypergraphs to verify rule-based expert systems," *IEEE Trans. on Knowledge and data engineering*, Vol.9, No.2, march-april, pp.221-237, 1997.

[19] C.L.Ramsey, L.B.Booker, "A parallel implementation of belief maintenance system," *AI Systems in Geverment Conference, Proceedings, Fifth Annual*, pp.180-186, 1990.

[20] M.Stanojevic, S.Vranes, D.Velasevic, "Using truth maintenance systems, A tutorial," *IEEE Expert*, Vol.9, No.6 pp.46-56, 1994.

[21] G.Valiente, "Verification of knowledge based redundancy and subsumption using graph transformations," *Int'l J. Expert System*, Vol.6, No.3, pp. 341-355, 1993.

[22] 박창현, 유석인, "지식 기반 시스템에서의 추론 Browser의 설계", *한국정보과학회논문지*, Vol.19, No.1, pp.80-93, 1992.



서 의 현

e-mail : ehsuh@mokwon.ac.kr

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, 에이전트 시스템