論文96-33B-6-10

FPN의 효율성 개선에 관한 연구

(A Study on Improving Efficiency of FPN)

任 在 傑 *, 李 桂 英 *, 李 太 庚 *

(Yim Jaegeol, Lee Gyeyoung, and Lee Taekyung)

요 약

본 논문은 이미 발표된 퍼지 페트리 네트를 이용한 추론 방법 중, FPN(Fuzzy Petri Net)[1] 방법의 효율성을 속도와 정확도 두가지 면에서 제고시키는 방법을 제안한다. 제안된 방법은 목적 명제에 대한 조건 명제의 관계에 있는 명제들 가운데 기초적인 명제를 사용한다. 기초적인 명제는 FPN에서 원천 정점에 해당함으로, 원천 정점을 찾는 대수적 방법 또한 소개한다.

Abstract

This paper presents a new method with which we can improve the efficiency (both of speed and precision) of FPN[1]. Our method makes use of the basic propositions among the conditional propositions of a goal proposition. The basic propositions are represented as source vertices on an FPN. Therefore, we introduce an efficient algebraic algorithm which finds source vertices of FPN.

I. 서 론

페트리 네트는 여러 사건이 동시에 발생하는 현상을 모형화하고 분석하는 가시적이며 수학적인 도구로써 근래에 들어 활발히 연구되고 있다.[1,2] 현재, 페트리 네트 자체에 대한 연구로는 고급 수준의 페트리 네트의 교착상태를 찾는 알고리듬[3]과 페트리 네트의 구조를 분석하는 대수적 방법[4]에 대한 연구가 진행되어 왔으며, 네트워크[5] 또는 인공지능[6] 등의 다른 분야에 페트리 네트의 이론을 적용하려는 실용적인 연구도 이루어지고 있다.

특히, 인공지능 분야에서는 인간의 지식 표현을 자연스럽게 표현하는 방법에 대한 연구 중에서 특히, 불분명한 지식을 표현하는 이론으로 주목받고 있는 퍼지 논리가 등장하였으며, 이러한 퍼지 논리와 페트리 네트의 접목에 대한 연구가 [1,2]와 [7]에 소개되었다. 본 논문에서는 [1,2]에 소개된 퍼지 페트리 네트 (FPN : Fuzzy Petri Net) 추론의 효율성을 제고시키는 다음과 같은 두가지 방법을 제안한다.

첫째, 사용자에게 이웃 장소에 해당하는 어떤 명제 d_i의 확신도(certainty factor)를 대화식으로 질의하는 방법에 대해서, 그것으로 d_i의 확신도를 계산해 낼 수 있는 근본적인 명제에 대한 확신도를 질의하도록 개선한다.

둘째, d_i의 확신도가 주어졌을 경우, 이로부터 목적 명제(goal proposition)d_j의 확신도를 추론하는 방법 대신에, 목적 명제 d_j만 주어지면 이의 확신도를 구하는 데 필요한 모든 근본적인 명제들의 확신도를 문의하여 d_j의 정확한 확신도를 구하는 방법을 제시한다. 이를 위하여 유향그래프에서 주어진 정점으로 들어오는 경로의 원천 정점과 이러한 경로상의 닫힌회로를 찾는 대수적 방법을 고안하여 사용한다.

* 正會員, 東國大學校 電子計算學科

(Dept. of Computer Science, Dongguk Univ.)

接受日字:1995年4月4日, 수정완료일:1996年5月20日

II. 퍼지 페트리 네트의 추론

규칙 기반 시스템을 위한 FPN[1,2]에 있어서, 퍼지 생성 규칙(fuzzy production rules)이란 여러개의 명

제들 사이의 퍼지 관계를 묘사하는 규칙을 일컫는다. 즉, 표 1과 같은 생성 규칙은 그림 1과 같은 FPN으로 표현되며, 그림에서 원으로 표현된 것을 장소(place)(p_i)라 하고, 막대로 표현된 것을 변천(transition)(t_i)이라 한다. 그리고, 장소들의 집합을 P, 변천들의 집합을 T라고 표기한다. 변천과 장소는 서로 유향 간선으로 연결되며, 유향 간선의 집합을 A(arc)라고 표기한다. 단, 변천과 변천, 장소와 장소가 유향 간선으로 연결되는 경우는 존재하지 않는다.

생성 규칙에 대한 FPN 모형을 구축하는 방법은 다음과 같다. 표 1과 같은 퍼지 생성 규칙이 주어지면 먼저 조건부의 명제가 or로 연결된 규칙들을 분리하고, 다음으로 각각의 생성 규칙에 대하여 변천을, 그리고 규칙에 사용된 명제 각각에 대하여 장소를 대응시킨다. 이때, 생성 규칙을 변천에 대응시킨 함수를 RT라 하고, 명제를 장소에 대응시킨 함수를 DP라 하고, 규칙 R_i의 조건부 명제들이 d_1, d_2, ...이고, 결론부의 명제들이 d_a, d_b, ... 일때, ($DP(d_1)$, $RT(R_i)$), ($DP(d_2)$, $RT(R_i)$), ...등과 ($RT(R_i)$, $DP(d_a)$), ($RT(R_i)$, $DP(d_b)$) ... 등의 유향 간선을 사용하여 변천과 장소들을 연결한다. 마지막으로 각 규칙의 확신도를 해당 변천에 기록하여 FPN으로 표현하게 된다.

유향 그래프에서 (v_i,v_j)가 유향 간선일 때, 정점 v_i는 정점 v_j의 입력 정점이라 하고, v_j는 v_i의 출력 정점이라 한다. 이와 마찬가지로 페트리 네트에서도 (p_i,t_j)가 유향 간선일 경우에 p_i는 t_j의 입력 장소라 하고, t_j는 p_i의 출력 변천이라 한다. 또한, (t_i,p_j)가 유향 간선일 때 t_i는 p_j의 입력 변천이라 하고, p_j는 t_i의 출력 장소라 한다. 어떤 장소(p_i)의 IRS(Immediately Reachable Set: 직접 도달 가능 장소)는 어떤 변천 t_k를 통하여 p_i와 연결된 장소들을 일컫는다.

$$IRS(p_i) = \{ p_j \mid \exists t_k \in T, (p_i, t_k) \in A \text{ and } (t_k, p_j) \in A \} .$$

장소 p_j가 $IRS(p_i)$의 원소이고, p_k가 $IRS(p_j)$의 원소이면, p_k는 p_i로 부터 도착가능(Reachable)하다라고 하며, 도착 가능한 관계는 직접 도착가능한 관계의 반사(reflexive) 추이(transitive) closure이다. 즉, 주어진 장소 p_i의 도착가능한 장소들의 집합을 $RS(p_i)$라 하면 $p_i \in RS(p_i)$이고, 만약 $p_k \in RS(p_i)$이면 $IRS(p_k) \subseteq RS(p_i)$가 된다.

[1]과 [2]에서는 두 개의 명제 d_s와 d_j가 주어질 때, 이들 사이에 관계가 있는지, 관계가 있다면 d_s의 확신 정도가 주어졌을 때, d_j의 확신 정도가 얼마나 되는지를 알아보는 퍼지 추론 방법이 제시되었다. 제시된 퍼지 추론 방법은 다음과 같이 요약된다.

표 1. 퍼지 생성 규칙의 예
Table 1. Example of fuzzy production rules.

R_1 : IF d_1 THEN d_2(CF = 0.85)
R_2 : IF d_2 THEN d_3(CF = 0.80)
R_3 : IF d_2 THEN d_4(CF = 0.80)
R_4 : IF d_4 THEN d_5(CF = 0.90)
R_5 : IF d_1 THEN d_6(CF = 0.90)
R_6 : IF d_6 THEN d_4 and d_9(CF = 0.95)
R_7 : IF d_1 and d_8 THEN d_7(CF = 0.90)
R_8 : IF d_7 THEN d_4(CF = 0.90)

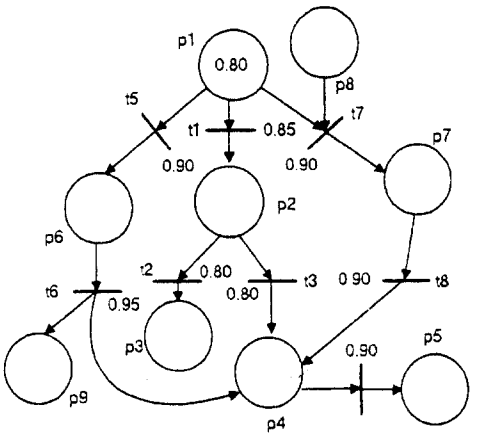

그림 1. 표 1의 퍼지 페트리 네트 표현
Fig. 1. An FPN representation for Table 1.

(1) d_s의 확신 정도를 d_s에 해당하는 페트리 네트의 장소(p_s)에 토큰(0 이상 1 이하의 실수)으로 표현하고, 이 장소의 출력 변천(t_i)들을 모두 격발한다. 입력 토큰의 최소값과 t_i의 확신도의 곱을 출력 장소의 토큰으로 놓는 것이 격발 규칙이다.

(2) t_i의 입력 장소 중 토큰을 갖고 있지 않은 장소(p_i)가 있을 경우에는, p_i에 해당하는 명제의 확신 정도를 사용자에게 대화식으로 질의한 후, p_i에 토큰을 놓아 t_i를 격발한다. 이 때, p_i와 같은 장소를 p_s의 이웃(adjacent)이라 한다.

(3) t_i의 격발 결과, t_i의 출력 장소는 새로운 토큰을 갖게 된다. 새로운 토큰을 얻은 t_i의 출력 장소 각각에 대하여 (1)과 (2)의 작업을 반복한다. 이러한 반복 작업을 p_s에서 시작하여 p_j에 연결되는 모든 경로상의 모든 장소에 대하여 반복 수행하며, 어떤 장소에 놓여지

는 토큰이 복수개일 때에는 확신도가 가장 큰 것을 그 장소의 토큰으로 한다.

예를 들어, 그림 1에서 명제 d_1의 확신도가 0.80이라고 주어지고, 명제 d_4의 확신도를 추론하는 문제의 경우에는 먼저 0.80을 p_1에 기입한 후, t_5와 t_1, t_7을 격발한다. 그리고, 변천 t_5의 격발 결과인 0.72는 p_6에, t_1의 격발 결과인 0.68을 p_2에 각각 놓는다.

다음으로 t_7을 격발하기 위하여 d_8의 확신도를 사용자에게 묻는다. 입력값이 0.70이라 가정할 경우, Min (0.80, 0.70) = 0.70을 구하여, 0.70과 0.90의 곱인 0.63을 p_7에 놓는다. 이와 같은 작업을 반복하면 p_4에 0.54, 0.68 그리고 0.57이 놓이게 된다. 결국, 이 중 가장 큰 값인 0.68을 d_4의 확신도로 출력하는 방법으로 귀결된다.

Ⅲ. 개선된 FPN 추론 방법

본 장에서는 기존의 추론 방법을 개선할 수 있는 방법을 제시한다. 페트리 네트를 표현하는 방법으로 발생 행렬(incidence matrix) [3] 을 사용하고 있다. 발생 행렬 A는 장소를 행으로, 변천을 열로 나타내는 |P|×|T| 행렬이며, (t_j,p_i)가 유향 간선이면 "A_{ij} = 1"로, (p_i,t_j)가 유향 간선이면 "A_{ij} = -1"로, p_i와 t_j가 직접 연결되어 있지 않으면 "A_{ij} = 0"으로 표현한다. 예를 들어, 그림 1의 FPN을 표현한 발생 행렬은 표 2와 같다.

표 2. 그림 1에 보인 FPN의 발생 행렬 표현
Table 2. The incidence matrix of the FPN shown in Fig. 1.

	t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8
p_1	-1	0	0	0	-1	0	-1	0
p_2	1	-1	-1	0	0	0	0	0
p_3	0	1	0	0	0	0	0	0
p_4	0	0	1	-1	0	1	0	1
p_5	0	0	0	1	0	0	0	0
p_6	0	0	0	0	1	-1	0	0
p_7	0	0	0	0	0	0	1	-1
p_8	0	0	0	0	0	0	-1	0
p_9	0	0	0	0	0	1	0	0

유향 그래프에서 입력 간선이 없는 정점을 원천정점(source vertex)이라 하고, 출력 간선이 없는 정점을 종말정점(sink vertex)이라 한다. 따라서, 장소 p_i는 발생 행렬의 i-th 행에 +1 항이 없을 때 원천 장소가 되고, -1 항이 없을 때 종말장소가 된다. 유향 그래프에서 정점들의 나열 v_1, v_2, ..., v_k는 2 이상 k 이하인 모든 i에 대하여 (v_{i-1},v_i)가 모두 유향 간선일 때, 경로라 한다. 또한 정점 v_k로 끝나는 경로를 v_k로 들어 오는 경로라 한다.

정점 v_k로 들어오는 극대 경로는 원천 장소에서 시작하는 v_k로 들어오는 경로이다. 따라서, 이때의 원천 장소를 v_k의 원천 장소라 한다. 그리고, 경로 중에서 처음 정점과 끝 정점이 같고, 나머지 정점들이 모두 유일할 때에는 닫힌 회로라 한다.

1. 효율적인 추론 알고리듬

기존의 추론 방법은 주어진 명제에 해당하는 장소 p_s로부터 주어진 목적 장소 p_j를 연결하는 통로상의 변천들을 격발해 나가는 과정에서, 어떤 변천이 토큰을 갖지 않는 입력 장소를 갖는 경우가 발생할 때에는, 이 입력 장소에 해당하는 명제의 확신도를 사용자에게 묻게 된다. 즉, 그림 1의 경우 d_1의 확신도가 0.8이라고 주어진 후, d_4의 확신도를 구할 때에는 t_7을 격발하게 되며, 이를 위하여 명제 d_8의 확신도를 입력하라고 사용자에게 요구하게 된다. 다음 예에서 보이는 바와 같이 이와 같은 사용자의 입력은 많은 시간을 소모한다.

예를들어 그림 1의 페트리 네트는 표 1에 보인 규칙들의 표현으로, 표 1에 다음과 같은 규칙 R_9을 더 첨가하면, 그림 1에 변천 t_9과 장소 p_{10} 그리고 p_{11}을 추가한 후, 유향간선 (p_{10},t_9), (p_{11},t_9), (t_9,p_8)로 이들을 연결하게 된다.

R_9: IF d_{10} AND d_{11} THEN d_8 (CF = 1.0),

이와 같이, 변형된 페트리 네트상에서 d_1의 확신도가 0.8일 때, d_4의 확신도를 구하려면 t_7을 격발하여야 하며, 위의 경우와 마찬가지로 d_8의 확신도를 사용자에게 질의하여야 한다. 그러나, 이 경우에 d_8은 규칙 R_9의 결론부의 명제로서, d_{10}과 d_{11}의 확신도를 바탕으로 R_9으로 부터 추론되어지는 명제이다.

실제 시스템의 경우에 d_8과 같은 결론부의 명제는 사용자가 일반적으로 알고 있는 평이한 지식이 아닌 경우가 많이 있다. 예를 들어, d_{10}은 "혈당량이 높다", d_{11}은 "아버지께서 당뇨가 있으셨다"와 같이 모든 환자의 검진표에서 직접 그 확신도를 알아 볼 수 있는 기본적 명제들인 반면에, d_8은 "당뇨 환자이다"와 같이 일반 사용자가 확신도를 알 수 없는 명제인 경우가 많다.

따라서, 추론 시스템은 결국 d_{10}과 d_{11}의 확신도를 다

시 문의하여야 한다. 사용자의 입력 속도는 컴퓨터의 처리 속도와 비교할 수 없을 만큼 느리다는 것은 주지의 사실이다. 더 나아가서 다음과 같은 규칙들이 더 첨가되는 경우에는 퍼지 페트리 네트 표현은 그림 2와 같이 변화되고, 사용자에게는 d_{11}, d_{12}, d_{13} 그리고 d_{14}의 확신도를 차례로 묻게 되며 결국, 추론에 걸리는 시간은 더욱 길어지게 된다.

R_{10} : IF d_{12} THEN d_{11} (CF = 0.9),

R_{11} : IF d_{13} THEN d_{12} (CF = 0.98),

R_{12} : IF d_{14} THEN d_{13} (CF = 0.95).

사용자의 입력을 여러번 요구하는 현상을 피하기 위해서는 d_8을 추론하는데 필요한 가장 기본적인 명제의 확신도를 질의할 수 있도록, 추론 방법을 개선할 필요가 있다. 즉, 페트리 네트에서 주어진 장소 p_i로부터 시작하여 입력 유향 간선을 역방향으로 추적하는 작업을 원천 장소에 도달할 때까지 반복하여야 한다. 이를 위해서는 먼저, 주어진 페트리 네트에 존재하는 유향 간선의 방향을 모두 역방향으로 바꾼 후, [알고리듬 1]에 보인 함수 find_source를 이용한다.

원천 장소는 발생 행렬상의 해당 행에 "+1" 엔트리가 없는 장소이다. 그리고, 변천 t_j가 주어진 장소 p_i의 입력 변천일 때에는 "$A_{ij} = 1$"이 되며, 그 역도 성립한다. p_i의 입력 변천을 찾기 위해서는 행렬 A의 i행에서 "+1" 엔트리를 찾으면 되고, 장소 p_i가 주어진 변천 t_j의 입력 장소일 때에는 "$A_{ij} = -1$"이 되고 그 역 또한 성립한다.

[알고리듬 1] 주어진 장소 p와 연결된 원천 장소를 찾는 알고리듬 : 페트리 네트의 유향 간선의 방향을 모두 역방향으로 바꾼 뒤 적용함.

```
find_source(p) {
    if p is a source then add p to source;
    else for each child C of p find_source(C);
}
```

따라서, t_j의 입력 장소는 행렬 A의 j열에서 "-1" 엔트리를 찾으면 되며, 장소 p_i의 원천 장소에 대한 검색은 p_i의 입력 변천을 찾고, 이들 변천의 입력 장소를 찾는 작업을 원천 장소를 만날 때까지 반복한다. 그러므로, 실제로는 역방향의 그래프를 그릴 필요가 없으며, [알고리듬 2]를 사용하여 발생 행렬상에서 원천 장소를 찾을 수 있다.

[알고리듬 2]는 문제의 p_i와 그의 원천 장소 사이에 닫힌 회로가 존재할 때, 무한 loop에 빠질 가능성이 있다. 따라서, 문제의 장소 p_i에 도달하는 이것의 원천 장소로 부터의 경로상에 닫힌 회로가 존재하는 가를 확인하여야 한다.

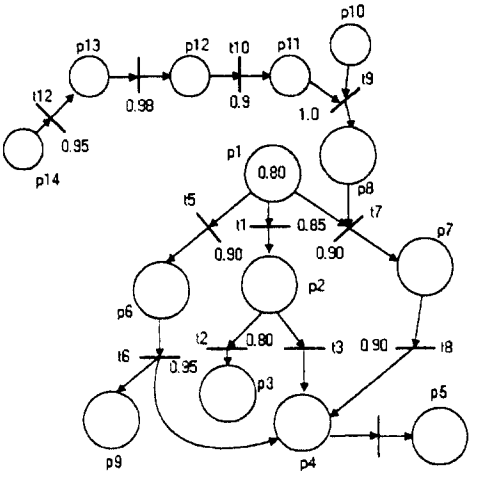

그림 2. 표 1에 몇 개의 규칙을 첨가한 경우의 FPN 모형

Fig. 2. An FPN model of production system consisting of the rules in Table 1 plus the rules shown in the above.

이를 위하여 본 논문에서는 주어진 페트리 네트를 구성하는 장소간의 의존성을 나타내는 dependency net를 구축하고, 이의 발생 행렬을 조작하여 문제의 장소 p_i의 원천 장소를 찾고, 이들을 잇는 경로상에 존재하는 닫힌 회로를 찾는 방법을 제시한다.

[알고리듬 2] 발생 행렬상에서 주어진 장소와 연결된 원천장소를 찾는 알고리듬.

```
/* node is an index of a transition or a place
*/
 find_source(node, p_or_t) {
  if (source(node, p_or_t)) {
     add_to_source(node); return();}
  else if (p_or_t == Place) {
    for(i=-1; i<=num_of_transitions; i++) {
      if(A [node] [i] == 1)
          find_source(i, Transition);
    }
  } /* else if */
  else
```

```
    for(i:=1; i<=num_of_places; i++)
      if(A [i] [node] == -1)
        find_source(i, Place);
}
```

주어진 페트리 네트의 장소간 dependency net는 주어진 페트리 네트의 장소들을 정점으로 하고, 페트리 네트상에 (p_i,t_k)와 (t_k,p_j)가 유향 간선인 변천 t_k가 존재하면 dependency net에 유향 간선 (p_i,p_j)를 삽입하여 구축한다. 예를 들어, 그림 1에 보인 페트리 네트의 장소간 dependency net는 그림 3과 같다.

장소간 dependency net를 컴퓨터에 표현하기 위한 발생 행렬을 정의한다. 장소간 dependency net의 정점의 수를 |P|라 할 때, 이의 발생 행렬 C는 |P|×|P| 행렬이며, 입력 발생 행렬 C^+에서 출력 발생 행렬 C^-를 뺀 나머지 $C = C^+ - C^-$로 정의된다. 여기서, C^+는 각 정점의 입력 유향 간선을 나타내는 행렬이고, C^-는 출력 유향 간선을 나타내는 행렬이다. 따라서, C^+의 일반항 C^+_{ij}는 다음과 같이 정의할 수 있다.

C^+_{ij} = 1 : iff (p_j, p_i)가 장소간 dependency net의 유향 간선,
= 0 : Otherwise.

그리고, 각 정점에서 출력되는 유향 간선을 나타내는 출력 발생 행렬 C^-의 일반항 C^-_{ij}는 다음과 같이 정의된다.

C^-_{ij} = 1 : iff (p_i, p_j)가 장소간 dependency net의 유향 간선,
= 0 : Otherwise.

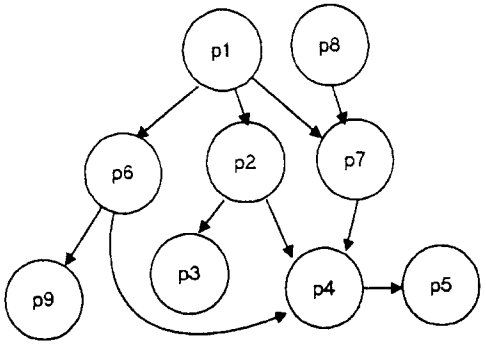

그림 3. <그림 1>에 대한 장소간 dependency net
Fig. 3. A place dependency net for the FPN shown in Fig. 1.

그림 3의 입력 발생 행렬 C^+와 출력 발생 행렬 C^-, 그리고 발생 행렬 C는 각각 표 3의 (a), (b), (c)에 보인 바와 같다.

여기서, C^+_{ij} = 1 이면 (p_j,p_i)가 dependency net의 유향 간선이고 그 역도 성립한다. 그리고, C^-_{ij} = 1이면 (p_i,p_j)가 dependency net의 유향 간선이고 그 역도 성립한다. 따라서, C^+_{ij} = 1이고 C^-_{ij} = 1이면, p_i와 p_j는 닫힌 회로가 되며, 이러한 닫힌 회로를 self-loop라고 한다. 그러나, 이 경우에 dependency net의 발생 행렬의 C_{ij}값은 C_{ij} = 1 - 1이 되어 0이 됨으로, p_i와 p_j가 직접 연결되지 아니한 경우와 self-loop를 이루는 경우를 발생 행렬 상에서는 구별할 수가 없게 된다. 그러므로, self-loop는 $C^+ + C^- = C$를 구하는 과정에서 찾아내어야 한다.

페트리 네트의 장소간 dependency net에 대한 입력 발생 행렬 C^-와 출력 발생 행렬 C^-도 페트리 네트의 발생 행렬 A로부터 직접 구할 수 있다. 정의에 따라, C'_{ij} = 1 iff (p_j,p_i)가 dependency net의 유향 간선이 된다. 그런데, (p_j,p_i)가 dependency net의 유향 간선이면 페트리 네트에서 (p_j,t_k)와 (t_k,p_i)가 유향 간선인 변천 t_k가 존재하는 것이고, 그 역 또한 성립한다.

표 3. 그림 3의 발생 행렬
Table 3. The incidence matrices C+, C- and C of the net shown in Fig. 3.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	0	0	0	0	0	0	0	0	0
p_2	1	0	0	0	0	0	0	0	0
p_3	0	1	0	0	0	0	0	0	0
p_4	0	1	0	0	0	1	1	0	0
p_5	0	0	0	1	0	0	0	0	0
p_6	1	0	0	0	0	0	0	0	0
p_7	1	0	0	0	0	0	0	1	0
p_8	0	0	0	0	0	0	0	0	0
p_9	0	0	0	0	0	1	0	0	0

(a) 입력 발생 행렬 C+.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	0	1	0	0	0	1	1	0	0
p_2	0	0	1	1	0	0	0	0	0
p_3	0	0	0	0	0	0	0	0	0
p_4	0	0	0	0	1	1	1	0	0
p_5	0	0	0	0	0	0	0	0	0
p_6	0	0	0	1	0	0	0	0	1
p_7	0	0	0	1	0	0	0	1	0
p_8	0	0	0	0	0	0	1	0	0
p_9	0	0	0	0	0	1	0	0	0

(b) 출력 발생 행렬 C-.

	p_1	p_2	p_3	p_4	p_5	p_6	p_7	p_8	p_9
p_1	1	-1	0	0	0	-1	-1	0	0
p_2	0	0	-1	-1	0	0	0	0	0
p_3	0	1	0	0	0	0	0	0	0
p_4	0	1	0	0	-1	1	1	0	0
p_5	0	0	0	1	0	0	0	0	0
p_6	1	0	0	-1	0	0	0	0	-1
p_7	1	0	0	-1	0	0	0	1	0
p_8	0	0	0	0	0	0	-1	0	0
p_9	0	0	0	0	0	1	0	0	0

(c) 발생 행렬 C.

그리고, 페트리 네트에서 어떤 변천 t_k에 대하여 (p_j,t_k)와 (t_k,p_i)가 유향 간선이라면, 이 페트리 네트의 발생 행렬 A_{jk}는 "-1"이 되고, A_{ik}는 "+1"이 되며 그 역도 또한 성립한다. 그러므로, 모든 장소 p_i에 대하여 A의 i번째 행에서 "+1" 엔트리를 찾고, 그 위치가 A_{ik}이면 이 엔트리가 있는 k 열에서 "-1" 엔트리를 찾아, 이의 위치가 A_{jk}이면 C^-_{ij} = 1로 하여 주는 작업을 모든 장소에 대하여 수행함으로써 입력 발생 행렬을 구축할 수 있다.

이와 마찬가지 방법으로, 모든 장소 p_i에 대하여 A의 i번째 행에서 "-1" 엔트리를 찾아 이의 위치가 A_{ik}이면 이 엔트리가 있는 k 열에서 +1 엔트리를 찾고, 이 위치가 A_{jk}이면 C^-_{ij} = 1로 하여 주는 작업을 모든 장소에 대하여 수행함으로써 출력 발생 행렬을 구할 수 있다. 그리고, $C^+ - C^-$ 연산을 하여 발생 행렬 C를 구할 수 있음은 앞서 언급 한 바와 같다.

[알고리듬 3] 페트리 네트의 발생 행렬 A로 부터 장소간 dependency net의 입력(C^-), 출력(C^-) 발생 행렬을 직접 생성하는 알고리듬.

```
for(i:=1; i<=number_of_places; i++){
    for(k:=1; k<=number_of_transitions; k++)
        if(A [i] [k] == 1)
            for(j:=1; j<=number_of_places; j++)
                if(A [j] [k] == -1) C+ [i] [j] = 1;
        else if(A [i] [k] == -1)
            for(j:=1; j<=number_of_places; j++)
                if(A [j] [k] == 1) C- [i] [j] = 1;
}
```

그리고, 장소간 dependency net의 정점 i가 입력 유향 간선이 없는 원천 정점일 때에는 발생 행렬의 i 행에 "+1" 엔트리가 없게 되며, i 열에는 "-1" 엔트리가 존재하지 않는다. [알고리듬 3]은 주어진 페트리 네트의 발생행렬 A로부터 장소간 dependency net의 입력 발생 행렬 C^+와 출력 발생 행렬 C^-를 직접 구하는 알고리듬을 제시한 것이다.

<성질 1> 정점 i가 원천 정점이면, C_{ij} != 1 for 1<=j<=|P| and C_{ki} != -1 for 1<=k<=|P|가 되며, 그 역도 성립한다.

페트리 네트의 self-loop가 아닌 닫힌 회로는 dependency net에서도 닫힌 회로이고, 그 역도 역시 성립한다. 그런데, self-loop는 dependency net를 구축하는 과정에서 이미 모두 찾아 내었으므로, dependency net의 닫힌 회로만 찾으면, 문제가 되는 회로는 모두 찾게 된다. 여기에서 문제가 되는 닫힌 회로는 문제의 장소 p_i에서 끝나는 모든 경로상에 존재하는 닫힌 회로이다. 문제의 장소 p_i로 들어오는 유향 간선의 시작 정점 p_j는 dependency net의 C_{ij} = 1로 나타난다.

그리고, p_j로 들어오는 유향 간선의 시작 정점 p_k는 dependency net에서 C_{kj} = -1로 나타난다. 그러므로, i번째 행의 모든 "+1" 엔트리 각각에 대하여 이의 위치가 C_{ij}라 할 때, j열의 모든 "-1" 엔트리 각각에 대하여 이의 위치를 C_{kj}라 하면, p_k에서 p_j를 통하여 p_i로 가는 경로가 존재하는 것이 된다. 문제의 p_i에 대한 이러한 p_k를 찾는 것은, dependency net의 발생 행렬 C의 i 번째 행에 나타나는 "+1" 엔트리를 소거하는 행 벡타의 합연산으로 수행될 수 있다.

합연산의 결과로 얻은 행을 C의 끝에 덧붙이고, 또 다시 이 행의 "+1" 엔트리를 소거하는 C의 행 벡타와의 합연산을 수행하여, 끝에 덧붙이는 과정을 반복함으로써 p_i로 들어오는 경로상의 정점들을 모두 찾을 수 있다. 이러한 과정에서 i, j 그리고 k를 모두 기록해 두었다가, 이미 기록된 행이 다시 사용되는 일이 발생하면 이는 닫힌 회로의 발견이 되므로 done이라고 표시한다.

합의 결과가 원천 정점의 위치에 "+1" 엔트리를 가지면, 그 정점이 곧 찾고자 하는 원천 정점이된다. 또한, 더하여지는 행이 원천 정점에 해당할 때에도 원천 정점을 찾은 결과가 되며, 이때에는 합에 done이라고 표시한다. 이러한 작업을 수행하려면 더하여지는 행과 중간 정점을 기록해 두어야 한다.

C의 엔트리 "1"은 입력 유향 간선의 존재를 나타내

는 의미가 있으며, 우리의 목적은 유향간선을 역방향으로 추적하여 원천장소를 찾는 것이다. 따라서, 우리가 사용하는 덧셈의 값은 더하여지는 값이 1이면 합의 결과를 1로하고 나머지 경우에는 모두 0으로 한다.

표 4는 그림 4에 보인 간단한 dependency net에서 정점 4로 들어오는 경로의 원천 정점을 찾는 과정을 보인 것이다. 표 4에 보인 행렬 C의 첫번째 열에 "-1" 엔트리가 없으므로, 정점 1이 원천 정점임을 우선 표시하고, 4행의 엔트리 "1"을 소거하기 위하여 2행을 4행에 더하여 그 합을 6행으로 한다. 이 때, 4행의 엔트리 "1"의 위치가 3이었고, 이를 소거하기 위하여 2행을 더하였음을 기록한다. 따라서, 6행의 정점 부분은 "4 3 2"이고, C 부분은 "1 0 0 0 1", I 부분은 "0 1 1 1 0"이 된다.

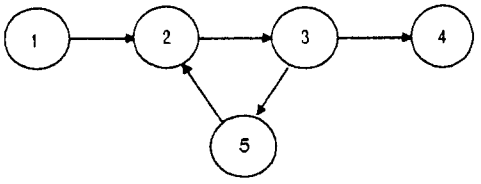

그림 4. 간단한 dependency net 예
Fig. 4. An example of a simple dependency net.

표 4. <그림 4>에서 정점 4의 원천 정점 찾기
Table 4. The process to find the source places of vertex 4 in Fig. 4.

	1 2 3 4 5	1 2 3 4 5
1	0 -1 0 0 0	1 0 0 0 0
2	1 0 -1 0 1	0 1 0 0 0
3	0 1 0 -1 -1	0 0 1 0 0
4	0 0 1 0 0	0 0 0 1 0
5	0 -1 1 0 0	0 0 0 0 1
4 3 2	1 0 0 0 1	0 1 1 1 0 --6
4 3 2 5 3	1 1 0 0 0	0 1 2 1 1 --7
정점부분	C 부분	I 부분

여기에서, 6행의 첫번째 엔트리가 "1"이므로 원천 장소 1이 2의 입력 장소이며, 또한 장소 1이 우리가 찾는 원천 장소임을 알 수 있다. 6행에는 엔트리 1이 1번 열과 5번 열에 존재하지만, 정점 1이 원천 정점이므로 5번 열의 엔트리 1만 소거하면 된다. 이를 위하여 5번 열의 엔트리가 "-1"인 3행을 6행에 더하고, 그 결과를 7번 행으로 한다. 이때 5번째 엔트리를 소거하였던 사실을 7행에 기록한다.

7행을 생성하는 데 관여한 정점은 4 3 2 5 3의 순서로 3이 두번 출현하므로 3 2 5 3이 닫힌 회로를 구성하고 있음을 알 수 있다. 그러나, 닫힌 회로를 찾는 작업을 새로운 행을 생성할 때 마다 매번 수행하는 것은 시간을 낭비한다. 따라서, 닫힌 회로의 존재 유무를 쉽게 알아보는 방법이 필요하게 된다.

표 4에 보이는 Identity matrix는 이를 위한 것으로 즉, 초기에 발생 행렬 C와 |P|×|P| identity matrix I를 병행하여 기록하고, i행과 j행을 k번째 "+1" 엔트리를 소거하기 위하여 더할 때, I의 i행과 j행의 합을 구하고 그의 k번째 엔트리에 1을 더하여 합을 구하는 데 관계된 모든 행들을 기록한다. 이때 I 부분의 합은 일반적인 덧셈 연산으로 구한다.

이때, I 부분의 j번 엔트리가 2이면 정점 j가 닫힌 회로에 속하게 되며, 이 닫힌 회로에 속하는 정점 중에서 목적 정점과 가장 가까운 것이 정점 j가 된다. 따라서, I 부분에 2가 엔트리로 출현할 때에만 닫힌 회로를 찾으면 된다. 주어진 목적 정점으로 들어오는 경로상에 존재하는 닫힌회로와, 들어오는 경로의 원천 장소를 찾는 알고리듬은 [알고리듬 4]와 같다.

[알고리듬 4] 주어진 목적 정점으로 들어오는 경로상에 존재하는 닫힌회로와 원천 장소를 찾는 알고리듬.
단계 1. [C]와 [I]를 병기한다.
단계 2. 원천 장소에 해당하는 열에 원천 열이라고 표시한다.
단계 3. 목적장소가 p_i이면 i행에 new라고 표시한다.
단계 4. new라고 표시된 행 중, 하나를 선택하여(i행이라 가정)

4.1) 원천열의 엔트리가 1이면, 이것을 p_i의 원천 장소로 기록한다.

4.2) +1 엔트리를 소거하여 새로운 행을 구한 후, new라 표시한다. 즉, i행 C부분의 +1 엔트리 중 원천 열이 아닌 것(k로 가정)에 대하여, C의 k열에서 -1 엔트리를 찾고(j행으로 가정), i행과 j행의 합을 구한다. 이때, C부분의 합 연산은 0+1=1+0=1+1=1, 이외의 경우는 0으로 계산한다. I부분은 보통의 산술 합산을 하며, i행의 정점 부분에 k, j를 덧부쳐서 새로운 행의 정점부분으로 한다.

4.3) 4.2에서 구한 행에서 닫힌 회로를 검색한다. 즉, I부분에 엔트리 2가 있으면 닫힌 회로가 발견된

경우이므로 new라는 표시를 삭제하고, 엔트리 2의 위치가 h라하면 정점 부분의 h에서 h까지의 부분 문자열의 역이 닫힌 회로가 된다.

[알고리듬 4]에서 발견되는 원천 장소는 목적 장소의 확신도를 추론하는데 관련된 가장 기본적인 명제들이 무엇인지를 알려준다. 그러므로, 이웃 장소에 해당하는 명제의 확신도를 묻는 대신 이들의 확신도를 문의하여 이웃 장소의 확신도를 추론하여야 한다. 그러나, 닫힌 회로상의 장소들 사이에는 어느 것이 더욱 기초 자료에 해당하는지의 관계가 존재하지 않는다.

따라서, 닫힌 회로가 발견되면 이것이 원천 정점에서 목적 정점까지의 어떤 경로와 공유하는 정점이 존재하는가를 검사하여, 그러한 정점이 존재할 경우에는 닫힌 회로상의 장소에 해당하는 명제의 확신도를 묻지 않고 [1]과 [2]의 알고리듬을 적용한다. 그러한 정점이 존재하지 않을 경우에는, 닫힌 회로상의 어떤 장소에 해당하는 명제의 확신도를 묻는 작업을 생략할 수가 없다. 닫힌 회로상의 명제들은 모두 같은 수준이므로, 목적 정점과 가까운 명제부터 확신도를 문의하면 된다.

2. 적용 및 검토

1) 추론 속도의 개선 문제

그림 2에서 명제 d_1의 확신도가 0.80일 때, 명제 d_4의 확신도를 찾는 문제의 경우, 기존의 방법은 t_7을 격발하기 위하여 d_8의 확신도를 묻고, 이의 확신도를 구하기 위하여 다시 d_{11}, d_{12}, d_{13} d_{14}의 순서대로 확신도를 묻게 된다. 그러나, 본 논문에서 제시되는 방법은 d_8의 확신도를 묻기 전에 p_8의 원천 장소들이 p_{10}과 p_{14}임을 [알고리듬 4]를 이용하여 발견해 내기 때문에, 처음부터 이들의 확신도를 문의하게 되므로, 사용자의 입력을 여러번 요구하는 문제를 해결할 수 있다.

다음은 닫힌 회로가 있는 경우의 적용 예로서, 이 때에는 다시 원천 장소의 유무에 따라 구분하여야 한다. 먼저, 원천 장소가 있는 <그림 5>의 경우, 명제 d_6의 확신도가 주어지고 명제 d_7의 확신도를 알고자 할 때, p_6의 이웃 장소가 p_4이므로 p_4의 확신도를 구하여야 한다. 이 때에는 p_4로 들어오는 경로상의 닫힌 회로와 원천 장소를 찾기 위하여 [알고리듬 4]를 이용하며, p_3, p_5, p_2, p_3가 닫힌 회로이고 p_1이 원천 장소이며, p_1에서 p_4에 이르는 경로가 p_1, p_2, p_3, p_4임을 찾는다. 그리고, 이들 간에는 공유하는 장소가 존재하므로, d_4의 확신도를 묻는 대신 d_1의 확신도를 물어, 기존의 추론 방법으로 d_4의 확신도를 계산하고 이의 결과를 이용하여 d_7의 확신도를 계산한다

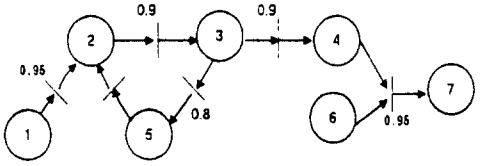

그림 5. 원천 장소와 닫힌 회로를 포함하는 dependency net

Fig. 5. A place dependency net including a cycle and a source place.

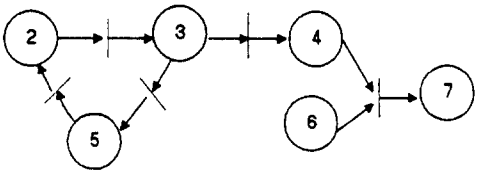

그림 6. 닫힌 회로를 포함하는 dependency net의 예

Fig. 6. An example of a place dependency net including a cycle.

목적 장소로 들어오는 경로상에 닫힌 회로가 존재하면서, 원천 장소가 없는 그림 6의 경우에는, d_4의 확신도를 구하기 위하여 역시 [알고리듬 4]로서 원천 장소와 닫힌 회로를 찾아 p_3, p_5, p_2, p_3가 닫힌 회로임을 알 수 있다. 이 때에는 닫힌 회로의 장소를 공유하는, 원천 장소로의 경로가 존재하지 않으므로, p_4와 가장 가까운 p_3에 해당하는 명제 d_3의 확신도를 대화식으로 묻게 된다.

2) 정확성 제고 문제

본 절에서는 기존의 추론 방법에서 무시하였던 부분을 고려하여 더욱 정확한 추론 결과를 얻는 방안을 제시한다. 표 1에 다음과 같은 규칙 R_9이 첨가되었다고 가정할 때,

R_9: IF d_{10} THEN d_4 (CF = 0.95)

이렇게 변화된 생성 규칙의 모형은 그림 7과 같이 표현된다.

d_1의 확신도가 0.8이라고 주어졌을 때, d_4의 확신도를 묻는 문제를 [1]의 방법으로 해결하게 되면, R_9에 해당하는 변천 t_9가 p_1과 p_4를 잇는 경로상에 있지 않으므로, 이 규칙은 d_4의 확신도를 구하는 데 전혀 영향

을 주지 못한다. 따라서, 앞서의 예와 동일한 결과를 얻게된다.

그러나, 만일 d_{10}의 확신도가 0.9였다면, t_9을 격발하여 얻는 p_4의 토큰은 0.855가 된다. 그리고, [1]에서와 같이 어떤 장소에 여러개의 토큰이 있을 때, 그 중 가장 큰 값을 취하는 공격적 추론을 채택하는 시스템에서는 0.855가 최종값이 되어야한다.

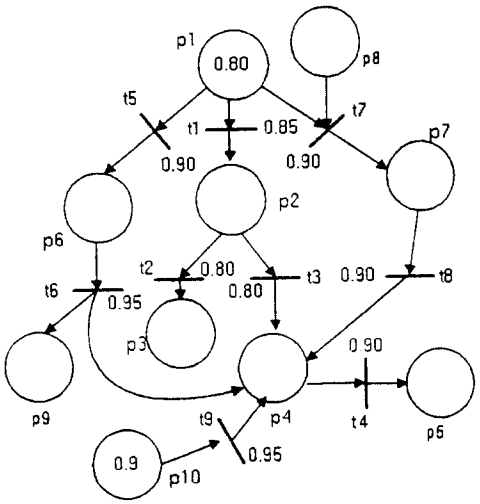

그림 7. p_s와 p_j를 잇는 경로 밖에 원천 명제가 존재하는 FPN의 예

Fig. 7. An FPN including a source place which does not lie on the paths from p_s to p_j.

이와 같이 d_{10}의 확신도가 d_4의 확신도를 추론하는 데 영향을 미치지 못하는 이유는, [1]의 추론 알고리듬이 주어진 d_s에 해당하는 장소(p_1)에서부터 d_j에 해당하는 장소(p_4)로 연결되는 경로상의 변천을 차례로 격발하여 나아갈 뿐, 그림 7의 p_{10}과 같이 이러한 경로상에 위치하지 않는 장소를 무시하기 때문이다.

이러한 문제를 해결하기 위하여,본 논문에서 제안한 추론 방법은 d_s와 d_j가 주어질 때, p_j의 모든 원천 장소(p_i)를 [알고리듬 4]를 적용하여 구한 후, p_i로부터 p_j까지의 경로상에 p_s를 포함하지 않는 모든 원천 장소에 해당하는 명제의 확신도를 문의한 후, 이들 각각으로부터 d_j의 확신도를 추론하는 것이다.

즉, 앞의 예에서는 d_{10}의 확신도를 문의하여, t_9을 격발하는 작업을 첨가하고, p_4의 토큰 중 최대값인 0.855를 d_4의 확신도로 추론하게 된다.

이렇게 수행하므로써 더욱더 정확한 추론이 이루어진다.

Ⅳ. 결 론

본 논문에서는 [1]과 [2]에 소개된 퍼지 페트리 네트의 추론 방법의 효율성을 제고 시키는 두 가지 방법을 제안하였다. 즉, 기존 방법의 문제점으로 지적될 수 있는 것으로는 사용자가 원시 데이타로부터 그 값을 알 수 있는 기본 명제의 확신도가 아닌 이웃 장소에 해당하는 명제의 확신도를 사용자에게 묻는다는 것과, 목적 명제와 조건부 관계가 있는 명제이더라도 주어진 명제에서 목적 명제에 이르는 경로상에 놓여있지 않으면 추론 과정에서 무시되는 것 등이 있다.

따라서, 본 연구에서는 첫째, 추론하여 나아가는 도중 어떤 명제 d_i의 확신도를 반드시 알아야 진행이 가능할 때, 기존의 방법이 직접 d_i의 확신도를 사용자에게 묻는데 반하여, d_i를 추론해 낼 수 있는 기초적 데이타 값에 해당하는 명제의 확신도를 물음으로써 추론 속도를 개선시키고자 하였다. 이를 위해서 본 연구에서는 관련 알고리듬을 새롭게 제시하였으며, 그 적용 효과를 고찰하였다.

둘째, 목적 명제 d_j가 주어지면 이를 추론하는 데 필요한 모든 기본 명제들에 대한 확신도를 물어 더욱 정확한 추론을 할 수 있도록 하는 방안을 제시하였다. 이러한 방법을 위해서 본 연구에서는 유향 그래프 상에 주어진 정점으로 들어오는 경로의 원천이 되는 장소와 이 경로상에 존재하는 닫힌 회로를 찾는 대수적 알고리듬을 제시하였으며, 이러한 방안이 퍼지 페트리 네트의 효율적인 추론 방법으로 효과적으로 적용될 수 있음을 보였다.

참 고 문 헌

[1] Shyi-ming Chen, Jyh-sheng Ke and Jin-fu Chang, "Knowledge Representation Using Fuzzy Petri Nets," IEEE Trans. on KDE, Vol. 2, No. 3, Sep., pp. 311-319, 1990.

[2] 조 상엽, 김 기태, "퍼지 페트리 네트를 이용한 퍼지 생성 규칙의 표현," 정보과학회논문지, 제21권 2호, 2월, pp. 298-306, 1994

[3] K. Barkaoui, C. von Bochmann, "An Efficient Algorithm for Finding Structural Deadlocks in Colored Petri Nets," Lecture Notes in Computer Science, Application and Theory of Petri Nets 1993, Vol. 691, pp.

69-88.

[4] 임 재걸, "장소-변천망의 구조적 특징 분석기 구현" 동국 논집 12집, pp. 157-178, 1993

[5] C. Capellmann and H. Dibold, "Petri Net Based Specifications of Services in an Intelligent Network - Experiences Gained from a Test Case Application," Lecture Notes in Computer Science, Application and Theory of Petri Nets 1993, Vol. 691, pp. 542-551.

[6] 임 재걸, "로봇 계획 문제에 Petri Net 도착 시험의 적용 방안" 동국 논총 32집, pp. 225-240, 1993

[7] C. G. Looney, "Fuzzy Petri Nets for Rule-based Decision making," IEEE Trans. Syst., Man, Cybern., Vol. SMC-18, No. 1, pp. 178-183, Jan./Feb. 1988.

저 자 소 개

任 在 傑(正會員)

1952년 7월 24일생. 1974년 인천교육대학 졸업. 1981년 동국대학교 전자계산학과 졸업(학사). 1987년 석사, 1990년 박사: University of Illinois at Chicago, Dept. of Electrical Engineering and Computer Science. 1974년 ~ 79년 초등교사. 1981년 ~ 81년 경제기획원 조사통계국 전산처리사. 1991년 ~ 92년 현대전자. 1992년 ~ 현재 동국대학교 전자계산학과(경주 캠퍼스) 근무, 부교수. 관심분야는 페트리 네트 응용과 인공지능

李 太 庚(正會員)

1980년 동국대학교 전자계산학과(학사). 1985년 숭실대학교 대학원 전자계산학과 (공학석사). 1994년 숭실대학교 대학원 전자계산학과 (공학박사). 1987년 ~ 현재 동국대학교 전자계산학과 부교수. 관심분야는 지식공학, 영상이해

李 桂 英(正會員)

1980년 동국대학교 전자계산학과 학사. 1983년 동국대학교 전자계산학과 석사. 1992년 단국대학교 전자공학과 박사. 1980년 Fujitsu Korea Ltd. 1985년 ~ 현재 동국대학교 전자계산학과 근무, 부교수. 1996년 ~ 현재 Washington State University, Visiting Professor. 관심분야는 자연어 처리, Speech processing, 패턴인식