論文95-32B-3-10

인공지능 생성시스템에서의 병렬 매칭

(A Parallel Matching in AI Production Systems)

姜 勝 一 *, 尹 鍾 旻 **, 鄭 圭 植 **

(Seung Il Kang, Jong Min Yoon, and Kyu Sik Chung)

요 약

생성시스템(Production System)을 실생활에 응용하는데 커다란 문제점중의 하나는 실행시간이 느리다는 점이다. 이러한 문제를 극복하기 위한 한가지 방법은 실행시간의 90% 이상을 차지하는 매칭시간을 단축하는 것이다. 본 논문에서는 다중 프로세서 환경에서 전형적인 매칭 알고리즘인 RETE 알고리즘을 병렬로 수행하여 매칭시간을 단축다. 분할전략은 규칙들 사이의 공통 조건수에 따른 규칙의 유사성을 평가하여 유사한 규칙들을 그룹으로 형성하기 위한 것이다. 할당 전략은 매칭 동작에서 요구되는 시간에 따라 분할된 그룹에 우선 순위를 정함으로써, 프로세서에 걸리는 부하가 균등하도록 하였다. 제안된 방법을 기존의 방법과 비교하기 위하여 OPS5 예제 프로그램을 사용하여 시뮬레이션을 수행하였다. 시뮬레이션 결과로부터 제안된 방법이 생성 시스템의 성능을 향상시킬 수 있음을 보였다.

Abstract

One of the hardest problems that limit real application of production system is its slowness. One way to overcome this problem is to speed up the matching operation which occupies more than 90% of the total execution time. In this paper, we try to speed up the matching operation with parallel execution of a typical pattern matching algorithm, RETE, in a multiprocessor environment. This requires not only to make partitions of the rules but also to allocate the partitioned rules to processors, respectively. A partition strategy is proposed to make groups of similar rules by evaluating the similarity of rules according to the number of common conditions between rules. An allocation strategy is proposed to make the load of each processor even by assigning the different priority to the group of rules according to the expected amount of time required for matching operation. To compare with the existing methods, we perform simulation using OPS5 sample programs. The simulation results show that the proposed methods can improve the performance of production system.

* 正會員, 大宇電子 SYSTEM 事業部
(System Development Team, DAEWOO Electronics Co.Ltd)

** 正會員, 崇實大學校 電子工學科
(Dep. of Elec. Eng., Soongsil Univ.)

接受日字 : 1994年 2月 3日

Ⅰ. 서 론

인공지능 기술을 응용한 시스템중의 하나인 전문가 시스템(Expert system)은 어떤 특정 분야의 지식을 컴퓨터에 옮겨서 전문가와 같은 능력을 발휘시키는 것을 목표로 하는 시스템이다. 이러한 전문가 시스템을 구축하는 주된 목적은 인공지능의 분야에서 얻어진 기술을 이용하여 지식형 시스템을 실현하고자 하는 것이다.

전문가의 지식을 컴퓨터에서 표현하는 기법들 중에서 대표적인 기법이 전문가의 지식을 규칙(rule)의 형태로 표현하는 규칙기반 시스템(rule-based system), 즉 생성 시스템[1]이다. 생성 시스템은 생성 메모리, 작업 메모리, 그리고 추론 엔진의 3가지 모듈로 구성되어 있다.

"생성 메모리(Production Memory 또는 규칙 메모리)"는 전문가의 지식들을 규칙의 형식(IF(조건부)THEN(실행부))으로 표현한다. 생성 메모리에 저장된 프로그램은 계산될 순서가 정해져 있지 않은 기본단위인 규칙(production rule)으로 구성되며, 이 규칙은 생성 시스템에서 실제로 수행할 계산을 나타낸다. "작업 메모리(Working Memory,데이타 메모리)"는 현재의 실세계의 상태 및 문제를 풀어 가는 계산과정중에 발생하는 중간 결과를 저장한다. "추론 엔진(Inference Engine)"은 생성 시스템 전체의 제어를 담당하는 가장 중요한 요소로서, 생성 메모리의 규칙과 작업 메모리의 데이타를 비교한 후에 선택된 규칙을 실행시키기 위해서 필요하다. 생성 시스템은 "추론 사이클(Inference Cycle 또는 recognize-act cycle)"이라 불리는 3단계 사이클로 수행된다. 첫번째, 패턴 매칭 사이클에서는 규칙의 조건부와 현재의 작업 메모리의 데이타와 비교하여 일치하는 규칙(들)을 찾아내어 그 규칙(들)을 충돌집합에 넣는다. 두번째, 충돌 해결 사이클에서는 충돌 집합내의 실행 가능한 여러 규칙들 중에서 실행하기 위한 하나의 규칙을 선택한다. 세번째, 규칙 실행 사이클에서는 선택된 규칙을 실행하여 현재의 작업 메모리를 수정한다.

다른 표현 기법들과 비교해 볼 때 생성 시스템은 전문가의 지식을 규칙기반 시스템의 IF-THEN의 제한된 구조로 쉽게 표현하고 조작할 수 있으며, 지식과 데이타를 분리하여 관리하기 때문에 규칙을 시스템에 추가하거나 삭제하기가 용이하다는 장점을 가지고 있다[1]. 반면에, 지식을 규칙이라는 구조적인 형태로 나타내야 하므로, 비구조적인 지식은 생성 시스템 모델과 같은 방법으로 표현할 수 없으며, 단순한 추론 엔진으로 순차적인 처리와 복잡한 순환과 같은 제어를 표현하기 어려운 단점을 가지고 있다. 특히, 생성시스템의 수행 속도가 순차 프로그램에 비하여 10배 또는 100배 정도 느리기 때문에 생성 시스템을 실시간에 응용하기에 어렵다는 문제점을 가지고 있다. 이러한 생성 시스템의 실행 시간을 단축하기 위한 여러 방법들의 연구가 진행되어 왔다. 지금까지의 연구 방향을 크게 세가지로 나누어 보면, 첫째로 데이터와 규칙의 비교횟수를 줄이거나 병렬 매칭 방법을 사용하여 효율적으로 네트워크를 운용하는 방식[2,4,6,8], 둘째로 OPS5 프로그램을 효율적으로 프로그래밍 함으로써 매칭시간을 단축하기 위한 방법[5], 그리고 셋째로 동시에 수행하여도 관계없는 규칙들을 찾아내어 동시에 수행함으로써 추론시간을 단축하는 방법[7,8,9]이다.

그 중에서 수행시간의 대부분을 차지하는 매칭을 효율적으로 수행하기 위한 대표적인 알고리즘이 RETE 알고리즘[2]이다. RETE 알고리즘은 매 사이클마다 모든 데이타와 규칙을 비교함으로써 발생되는 불필요한 매칭 과정을 줄이기 위해 규칙들 사이의 유사성 있는 조건을 공유하고 매사이클마다 중간 결과를 네트워크에 저장한다. 본 논문에서는 병렬환경에서 동시에 매칭을 수행함으로써 전체 추론시간을 단축하기 위한 방법으로 새로운 병렬 매칭 알고리즘을 제안하고자 한다. 즉, RETE 알고리즘을 병렬환경에서 효율적으로 수행하기 위하여, 네트워크 공유의 장점을 유지하고 RETE 알고리즘의 병렬성을 최대한 활용하여 병렬환경을 구축하기 위한 규칙의 분할(partition) 및 할당(allocation) 알고리즘을 제안하고자 한다. 분할시 고려사항은 네트워크 공유를 최대한 유지하기 위해 규칙을 분할할 때 규칙들의 조건부 사이의 유사성을 분석하여 조건을 최대한 공유할 수 있도록 규칙들을 분할한다. 이를 위해 본 논문에서는 OPS5 예제 프로그램에서 규칙들의 조건부를 비교한 후에, 공통조건수가 많은 규칙들을 같은 그룹으로 배정하는 분할 방법을 이용한다. 분할된 규칙의 집합들을 프로세서에 균등하게 할당하기 위한 고려사항으로 매칭 소요시간에서 차지하는 비중이 큰 요소부터 우선적으로 고려한다. 본 논문의 구성은 2장에서 생성시스템에서의 패턴 매칭을 설명하고, 3장에서 제안된 병렬 패턴 매칭 알고리즘을 설명하고, 4장에서는 시뮬레이션 수행 환경 및 제안된 방법의 수행결과에 대해 분석을 한다. 마지막으로, 5장의 결론으로 끝맺는다.

Ⅱ. 생성 시스템에서의 패턴 매칭

1. 생성 시스템에서의 패턴 매칭 알고리즘

생성 시스템의 패턴 매칭 알고리즘을 크게 상태저장

(state saving) 알고리즘과 비상태저장(non-state saving) 알고리즘[4]으로 나눌 수 있으며, 대부분의 생성 시스템에서는 특성상 상태저장 알고리즘을 사용한다. 비상태저장 알고리즘은 매 사이클마다 모든 규칙과 작업 메모리의 내용을 비교하는 반면에, 상태저장 알고리즘은 이전 사이클에서의 매칭 결과가 저장되어 있어서 매 사이클마다 가장 최근에 실행된 규칙에 의해 변화된 작업 메모리의 변화에 대해서만 매칭을 수행한다. RETE 알고리즘은 상태저장 알고리즘의 대표적인 것으로, 이 알고리즘의 구현은 컴파일 시에 각 규칙들 간의 유사성, 즉 공통된 조건부를 고려하여 조건부를 공유하는 트리구조로 된 네트워크를 구성한 후에, 이를 이용하여 매칭을 수행한다. 그리고 매칭된 중간 결과를 네트워크의 노드에 저장한다.

규칙의 조건부의 내용과 작업 메모리의 내용이 일치하는지 여부를 비교하는 것을 "매칭(matching)"이라 하고, 규칙의 조건부와 작업 메모리가 일치(full matching)하였을 때 "매칭이 성공되었다"고 한다. 규칙의 모든 조건이 작업 메모리와 만족되었을 경우에 이 실행가능한 규칙을 "규칙-데이타 쌍(Instantiation)"이라 하고, 이 규칙은 현재 사이클에서 실행 가능한 규칙들이 저장된 충돌 집합에 들어간다.

네트워크를 이용하여 매칭 동작을 수행하는 동작 원리는 다음과 같다. 규칙과 작업 메모리가 아래와 같은 경우를 생각해 보자.

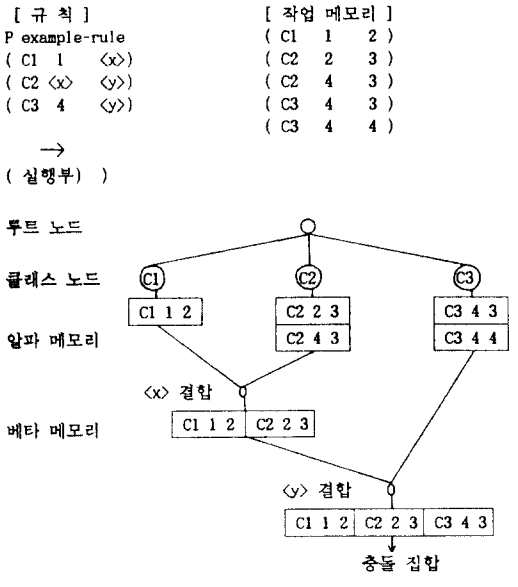


그림 1. RETE 네트워크
Fig. 1. RETE Network.

그림 1에 보인 바와 같이 5개의 작업 메모리 요소가 각각 토큰의 형태로 네트워크의 루트 노드에 전달된다. 입력이 한개인 "1-입력 노드(one-input node)"는 클래스와 상수 검사를 수행하고 매칭에 성공한 토큰은 알파 메모리에 저장된다. 예를 들면, 루트 노드에 전달된 첫번째 토큰 (C1 1 2)은 클래스 노드에서 클래스 이름을 검사한다. 이때, 클래스 노드와 클래스 이름 (C1)이 일치하는 토큰(C1 1 2)은 상수검사를 거친 후에 알파메모리에 저장된다. 나머지 토큰들이 차례로 클래스 노드 C1에 전달될 때 클래스 이름이 일치하지 않으므로, 다음 클래스 노드인 C2, C3에 차례로 전달되어 같은 방법으로 처리된다. 입력이 두개인 "2-입력 노드(two-input node)"는 두 조건간의 변수 결합(variable binding)을 수행하고, 결합이 성공한 토큰을 베타 메모리에 저장한다. 예를 들면, 알파 메모리에 저장되어 있는 첫번째 토큰(C1 1 2), 두번째 토큰(C2 2 3)은 규칙의 첫번째와 두번째 조건에서 공통변수(〈x〉)의 값이 2로 일치하므로, 더 큰 토큰((C1 1 2)(C2 2 3))을 형성한 후에 베타 메모리에 저장된다. 같은 방법으로 공통 변수 〈y〉에 대해 마지막 변수 결합이 성공한 토큰 ((C1 1 2)(C2 2 3)(C3 4 3))을 규칙-데이타 쌍이라 하며 충돌 집합에 저장한다[2]. 이러한 RETE 알고리즘은 단일 프로세서 하에서 계산의 반복을 피하기 위해 네트워크를 공유하는 점에서는 장점이 될 수 있으나 병렬 매칭 하에서는 프로세서간의 통신 문제를 야기시킨다.

2. 기존의 병렬 매칭 알고리즘

생성 시스템을 병렬환경에서 수행할 경우 여러 레벨의 병렬성을 활용할 수 있도록 병렬 수행 모델이 제안되었다.[6] 이 모델에서는 크게 3가지 레벨의 병렬성 즉, 개념 레벨 병렬성(concept level parallelism), 단계간 병렬성(inter-phase level parallelism), 단계내 병렬성(intra-phase level parallelism)을 활용할 수 있다. 병렬 수행 모델을 매칭에 국한하여 살펴보면 단계내 병렬성이 관련된다.

제안된 모델에서 단계내 병렬성[4,6]은 RETE 네트워크 분할의 granularity 정도에 따라 노드 레벨 병렬성, 규칙 레벨 병렬성, 조건 레벨 병렬성의 3가지로 나눌 수 있다.

노드 레벨 병렬성(node level parallelism)[3,4,6]은 네트워크에서 다른 노드의 활성화를 허용함으로써 이루어지며, 이는 토큰이 노드의 입력에 도착할 때마다 노드 활성화에 관한 일을 수행할 새로운 처리과정을 만듦으로서 간단히 구현될 수 있다. 규칙 레벨 병렬성(production level parallelism)[6]은 모든 규칙을 동시에 매칭하는 개념으로 한 규칙내의 모든 처리는

순차적으로 실행된다. 각 규칙에 대한 매칭을 수행하는 과정에서 프로세서간의 통신이 필요 없다는 장점이 있으나, 네트워크의 노드가 공유되지 않기 때문에 단일 프로세서 하에서 보다 매칭 단계에서 필요 이상의 계산을 수행하게 되는 단점이 있다. 조건 레벨 병렬성(condition level parallelism)[16]은 노드 레벨의 병렬성과 규칙 레벨의 병렬성의 대안으로서, 조건 요소내의 모든 상수 검사가 단일 처리과정에 의해 수행되며 변수 검사는 분리된 처리과정에 의해 수행된다. 본 논문에서는 규칙레벨의 병렬성을 활용하여 RETE 네트워크를 이용한 병렬 매칭 알고리즘을 제안하고자 한다.

Ⅲ. 병렬 패턴 매칭 알고리즘

본 논문에서는 실행 시간의 대부분을 차지하는 매칭 단계에 한정하여 규칙 레벨 병렬성을 구현하고자 한다. 즉, RETE 네트워크를 규칙 레벨에서 분할하고 다중 프로세서에 할당하여 수행한다. 알고리즘 제안시 고려한 사항은 병렬 환경 하에서 일부 네트워크를 공유함으로써 발생되는 프로세서간의 통신 부담을 줄이고, 규칙 레벨 병렬성 수행시에 단일 프로세서 환경에서 매칭 수행 과정에서보다 필요 이상의 계산을 수행하지 않도록 하였다.

이를 위해 본 논문에서는 규칙 레벨 병렬성을 적용할 경우에 규칙들의 조건부사이의 유사성을 고려하여 공통 조건을 최대한 공유하도록 규칙들을 분할하여 그룹을 형성한다. 즉, 규칙의 조건부를 비교하여 최대 공통 조건수가 큰 규칙들을 동일 그룹내에 속하도록 규칙들을 분할한다. 형성된 그룹들 사이에서 규칙의 조건부의 변수의 수, 속성-값 쌍의 수, 조건의 수와 규칙의 수를 고려하여, 분할된 규칙들을 각 프로세서에 균등하게 할당한다. 구체적인 분할 및 할당 알고리즘은 다음과 같다.

1. 병렬 매칭을 위한 분할 알고리즘

규칙을 그룹으로 분할하는 기본 개념은 크게 4단계로 나눌 수 있다. 첫번째 단계에서는 같은 그룹에 포함시킬 수 있는 규칙 즉, 한 규칙의 조건부의 내용이 다른 규칙의 조건부의 내용에 포함되는 규칙들을 우선 찾아낸다. 두번째 단계에서는 규칙들 사이의 최대 공통 조건수를 조사하여 새로운 그룹을 생성하거나, 또는 기존 그룹에 병합하거나 또는 그룹배치를 보류한다. 그룹배치를 보류할 경우는 한 규칙이 두개 이상의 그룹에 속할 경우로 현 단계에서는 그룹 배치를 보류하고, 현 단계에서 만들어진 그룹의 공통조건이 기존 그룹의 공통 조건에 포함될 경우에 기존 그룹에 병합시키고, 포함되지 않을 경우에 새 그룹을 형성한다. 세번째 단계에서는 그룹 배치가 보류된 규칙들을 관련된 그룹의 특성을 고려하여 균등하게 그룹을 배치한다. 네번째 단계에서는 공통 조건이 없는 규칙들을 나머지 그룹으로 형성한다. 이러한 과정을 순서도로 표시하면 그림 2와 같다.

프로그램의 규칙의 수는 m개이고, 임의 규칙(Ri) 조건부의 조건이 $C_{i1}, C_{i2}, \ldots, C_{is}$이고, 규칙(R_j) 조건부의 조건이 $C_{j1}, C_{j2}, \ldots, C_{jt}$로 구성되며, 형성되어질 그룹을 $G_1, G_2, \ldots, G_h$로 가정한다.(h ≤ m)

규칙_집합(R) : $R = \{ R_1, R_2, R_3, \ldots R_m \}$

분할_그룹(G) : $G_1, G_2, G_3, \ldots G_h$ (h < m, m: 규칙의 수)

공통_조건수(k) : k ≤ n (n : 최대 공통조건수)

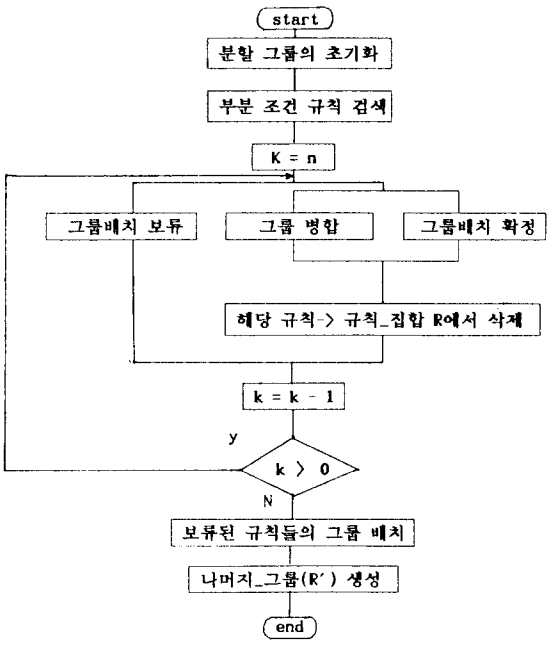


그림 2. 분할 알고리즘 순서도
Fig. 2. Flow Chart of Partition Algorithm.

본 논문에서 제안된 분할 알고리즘을 의사 코드(pseudo code)를 이용하여 자세히 설명하면 다음과 같다.

```
선언부 ;                    /* 초 기 화 */
 규칙_집합 (R) : m개의 규칙의 리스트 ;
   { R = {R1, R2, .., Rm} }
 규칙_조건 (C) : m개 규칙 조건부의 조건 리스트 ;
   { C = { C11, C12, ..., C1s, .., Cj1, Cj2, .., Cjt, ... } }
```

분할_그룹 (G_h) : m개의 규칙이 분할되어질 h개의 그룹 리스트 ;

{ $G_1, G_2, \ldots G_u, G_v \ldots G_h$ } 단, ($h \le m$)

{ $G_i = \{C_i, F\}$ (C_i:그룹 G_i의 공통조건, F:플래그 }

공통_조건수 (k) : 규칙들의 조건부들 사이의 공통조건수(k=n) ;

임의_규칙(R_i)_조건부(C_{ip}) : ($1 \le p \le s$)

{ 임의의 한 규칙 Ri의 조건부의 조건의 리스트 }

임의_규칙(R_j)_조건부(C_{jq}) : ($1 \le q \le t$)

{ 임의의 한 규칙 R_j의 조건부의 조건의 리스트 }

나머지_그룹(R') : 마지막 단계에서 규칙_집합 R에 남아있는 규칙 리스트

플래그(F) : { F = { 'D','I','M','U'}

D: 그룹 배치 확정, I: 그룹 배치 보류

M: 그룹 병합, U: 그룹배치 미결정 }

시작

단계 1. 각 규칙 R_i가 각 그룹 G_i에 속하도록 분할_그룹(G_h)을 초기화한다.

{ G_i = { C_i, 'U' }

C_i:그룹 Gi의 공통조건을 규칙 R_i의 조건부로 초기화, F = 'U' }

단계 2. 규칙_집합 R에서,

{ 임의_규칙(R_i)_조건부(C_{ip}) ⊂ 임의_규칙(R_j)_조건부(C_{jq}) 이면,

⇒ 규칙 Ri를 규칙 R_j가 속하는 그룹에 포함시킨다.

⇒ 규칙 R_i를 규칙_집합 R에서 삭제한다.

⇒ 규칙 R_i가 속한 분할 그룹(G_i)의 플래그를 'M'으로 설정한다.

⇒ 규칙 R_j가 속한 분할 그룹(G_j)의 플래그를 'D'로 설정한다. }

단계 3. 규칙_집합 R에서,

루프

{ 공통_조건수 k=n 부터 k=1 까지에 대해,

⇒ 공통_조건수 k를 하나씩 감소시키면서 다음 과정을 반복한다. }

{ 2.1 공통_조건수가 k개인 규칙들이 있으면,

⇒ 다음 세가지중 한가지 실행한다.

2) 그룹배치 보류 :

{어떤 규칙 Ri가 두개이상의 그룹에 속할시 (($R_i \subset G_u$)∧($R_i \subset G_v$))

⇒ 규칙 R_i가 속한 분할 그룹(G_i)의 플래그를 'I'로 설정한다. }

3) 그룹 병합 ;

{ 그룹(G_i)의 공통조건(C_i) ⊂ 그룹(G_j)의 공통조건(C_j)

⇒ 규칙 R_i가 속한 분할 그룹(G_i)의 플래그를 'M'으로 설정한다.

⇒ 규칙 R_j가 속한 분할 그룹(G_j)의 플래그를 'D'로 설정한다. }

1) 기존_그룹 유지 :

{ 그룹(G_i)의 공통조건(C_i) ⊄ 그룹(G_j)의 공통조건(C_j)

⇒ 규칙 R_i가 속한 분할 그룹(G_i)의 플래그를 'D'로 설정한다.

⇒ 각 경우에 있어서 규칙 R_i를 규칙_집합 R에서 제거한다. }

루프 종료 ;

단계 4. 규칙_집합 R에서,

{ 규칙 R_i가 속한 분할 그룹(G_i)의 플래그가 'I'인 경우에,

⇒ 아래 방법대로 관련된 그룹들 중에서 한 그룹에 배치한다. }

1) 관련된 그룹들의 변수의 수가 균등하도록 규칙을 배치한다.

2) 관련된 그룹들의 속성-값 쌍의 수가 균등하도록 규칙을 배치한다.

⇒ 규칙 R_i가 속한 분할 그룹(G_i)의 플래그를 'M'으로 설정한다. }

단계 5 규칙_집합 R에서,

{ 규칙 R_i가 속한 분할 그룹(G_i)의 플래그가 'U'인 경우에,

⇒ 나머지_그룹(R')생성한다. }

종료.

본 논문에서 제안된 분할전략은 RETE 알고리즘의 장점인 네트워크의 공유를 최대한 유지하기 위하여 규칙들 간의 공통조건을 중심으로 작성하였다. MAB (Monkey and Banana)프로그램을 예를 들어 살펴보면 단계 2에서 한 규칙의 조건부가 다른 규칙의 조건부의 부분 조건인 경우는 Mb0(C1, C2), Mb1(C1, C2, C3) ⊂ Mb2(C1, C2, C3, C4)로 규칙 Mb0와 Mb1이 속한 분할 그룹(G_0, G_1)을 규칙 Mb2가 속하는 그룹(G_2)에 포함시킨다. 단계 3의 세가지 경우에 대해 구체적으로 살펴보면 다음과 같다. 첫째, 그룹 배치가 보류된 경우('I')의 예를 들면 그룹 G_u 공통조건

이 C1, C2이고 그룹 G_v 공통조건이 C2, C3일 때 규칙 Ri의 조건부가 (C1, C2, C3)으로 구성될 경우에 Ri의 그룹배치를 보류한다. 둘째, 그룹을 병합하는 경우의 예를 들면 공통 조건수가 k가 3일때 그룹 G_u 공통조건이 C1, C2, C3이고 공통 조건수 k가 2일 때 그룹 G_v의 공통조건이 C1, C2일 때(공통조건 : { C1,C2 } ⊂ { C1,C2,C3 }) 그룹 G_v를 그룹 G_u에 병합한다. 셋째, 그룹 배치를 확정하는 경우는 그룹(G_i)의 공통조건이 그룹(G_i)의 공통조건에 포함되지 않는 경우이다.

2. 병렬 매칭을 위한 할당 알고리즘

분할된 h개의 그룹을 i개의 프로세서에 균등하게 할당하기 위하여, 매칭 수행시간에 차지하는 비중이 큰 요소들을 우선적으로 고려하여 4단계로 나누어 수행했다. 첫번째 단계에서 나머지 그룹을 제외한 h-1개의 그룹에서 변수의 수가 균등하게 프로세서들에 할당한 후 나머지 그룹을 해체하여 각 프로세서에게 같은 방식으로 수행하고 나머지 세단계에서 각 그룹내의 속성-값 쌍의 수, 조건의 수, 규칙의 수를 고려하여 분배하였다. 이 과정을 순서도를 이용하여 설명하면 그림 3과 같다.

분할된 그룹 집합 (G_i) : $G_1, G_2, G_3 \ldots G_h$ $(1 \le i \le h)$

그룹_조합(C) : $C_1, C_2, C_3 \ldots C_i$

프로세서 수 : i개

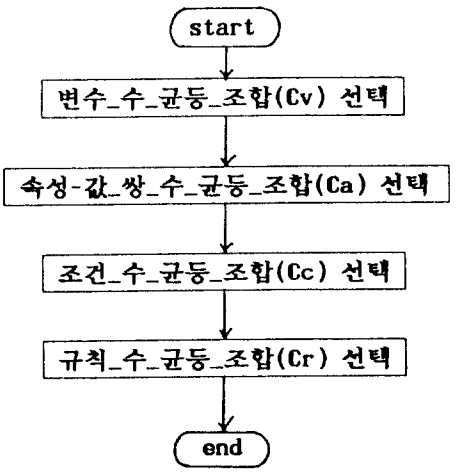


그림 3. 할당 알고리즘 순서도
Fig. 3. Flow chart of Allocation Algorithm

본 논문에서 제안된 할당 알고리즘을 의사 코드를 이용하여 자세히 설명하면 다음과 같다.

선언부 /* 초기화 */

분할_그룹(G_h) : m개의 규칙이 분할된 h개의 그룹 리스트 ;

{ $G_1, G_2, G_3, \ldots G_h$ } 단, $(h \le m)$

변수_수_균등_조합(Cv) : 변수의 수가 균등한 조합의 리스트 ;

속성-값_쌍_수_균등_조합(Ca) : 속성-값 쌍의 수가 균등한 조합의 리스트 ;

조건_수_균등_조합(Cc) : 조건의 수가 균등한 조합의 리스트 ;

규칙_수_균등_조합(Cr) : 규칙의 수가 균등한 조합의 리스트 ;

시작

단계 1 h개의 분할_그룹(G_h)들에 대해 :

{ 나머지_그룹(R')을 제외한 h-1개의 그룹들에 대해,

⇒ 가능한 모든 그룹의 조합을 만든다.

⇒ 모든 조합의 변수의 수가 균등해지도록 나머지_그룹(R')에서 변수를 포함한 규칙을 분배한다.

⇒ 변수_수_균등_조합(Cv)들을 선택한다. }

단계 2 변수_수_균등_조합(Cv)들에 대해 :

{ 속성-값 쌍의 수가 균등하도록,

⇒ 나머지_그룹의 규칙을 분배한다.

⇒ 속성-값_쌍_수_균등_조합(Ca)들을 선택한다.

}

단계 3 속성-값_쌍_수_균등_조합(Ca)들에 대해 :

{ ⇒ 조건_수_균등_조합(Cc)들을 선택한다

⇒ 없을 경우, 조건의 수의 차가 가장 적은 조합을 택한다. }

단계 4 조건_수_균등_조합(Cc)들에 대해 :

{ ⇒ 규칙_수_균등_조합(Cr)을 선택한다

⇒ 없을 경우, 규칙의 수의 차가 가장 적은 조합을 선택한다. }

종료.

본 논문에서 제안된 할당전략은 매칭 수행시간에서 차지하는 비중이 큰 부분을 우선적으로 고려하였다. 첫번째로 변수의 수를 고려한 근거는 매칭수행시 변수 결합과 변수 검사에 관련된 사항이 전체 매칭 수행과정중 80%이상을 차지하기 때문이다. 두번째로 속성-값 쌍의 수는 네트워크 형성시 네트워크의 깊이에 비례하며 네트워크의 깊이가 깊을수록 일반적으로 토큰이 충돌집합까지 이르는데 시간이 많이 걸린다. 세번째 고려사항인 조건의 수는 네트워크의 너비와 관련이 있으며 루트노드에서 다음노드에 뿌려줄 토큰의 수에 비

례한다.

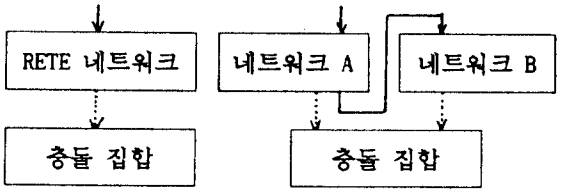
네번째 고려사항인 규칙의 수는 매칭이 완료된 후에 충돌해결 및 규칙실행 과정에 관련된 부분이다.

Ⅳ. 시뮬레이션

1. 시뮬레이션 환경

본 논문에서 제안된 병렬 매칭 알고리즘의 성능을 평가하기 위하여 SUN4/SLC 워크스테이션에서 생성 시스템 언어인 OPS5를 중심으로 시뮬레이션을 수행하였다. 본 시뮬레이션을 위해 사용된 OPS5 번역기는 LISP 언어로 작성된 번역기로서 이 번역기를 실행하기 위해 LISP 인터프리터를 사용하였다. 기존의 OPS5 인터프리터는 단일 프로세서 환경 하에서 RETE 알고리즘을 기반으로 작성되었다. 그러므로 다중 프로세서 환경 하에서 병렬 매칭 알고리즘을 수행하기 위해 OPS5 인터프리터를 변형하여 기존의 단일 매칭으로 수행되던 매칭 과정을 여러 개의 부분매칭으로 나누어 각 부분매칭에서 수행시간을 측정하여 그 중에서 가장 긴 수행시간이 그 병렬 매칭 사이클에서의 전체 매칭 시간으로 간주하는 방식으로 전 사이클에서 소요되는 매칭 수행시간을 계산하였다. 이 수행시간을 기존의 단일 매칭 방법에서의 전체 수행시간과 비교하였다.

본 논문에서 시뮬레이션의 간편성을 위하여 프로세서가 2개인 병렬 수행환경이라는 가정 하에서 시뮬레이션을 수행하였다. 즉, 기존의 단일 네트워크를 2개의 네트워크인 네트워크 A와 B로 나누어 각각의 네트워크를 이용하여 병렬 매칭을 수행하였으며, 병렬 매칭과정에서 작업 메모리와 충돌집합은 두 네트워크에서 공유하도록 하였다. 병렬 매칭 수행시간을 측정하는 방법은 네트워크 A와 B를 이용하여 순차적으로 매칭을 수행하고, 두 매칭과정에서 많은 시간을 소비한 네크워크의 수행시간으로 계산하였다.



(a) 순차 매칭 흐름도 (b) 병렬 매칭 흐름도

토큰 흐름 : ——〉

규칙-데이타 쌍의 흐름 : ……〉

그림 4. 패턴 매칭 구성도

Fig. 4. Diagram of pattern matching.

2. 시뮬레이션 조건 및 결과

5개의 OPS5 예제 프로그램인 MAB(원숭이와 바나나 문제), NQUEEN(서양 장기), Mincut(회로 설계 문제), Toru(Toru에 의해 수정된 Waltz 문제), Tourney(다리관광 스케줄링 문제) 프로그램이 시뮬레이션을 위해 사용하였다. 표 1은 각 OPS5 예제 프로그램들의 프로그램 특성 및 순차 매칭 시간을 나타낸다.

표 1. OPS5 예제 프로그램의 특성 및 순차 매칭 시간

Table 1. Characteristics of OPS5 example Programs and their execution time of sequential matching.

프로그램	M A B	Nqueen	Toru	Mincut	Tourney
규칙의 수	20	19	48	43	17
조건의 수	45(24)	68(49)	191(75)	131(84)	80(49)
사이클 수	16	27	212	539	528
전체 시간1	1011	2820	36921	211362	3256525
매칭 시간1	706	2115	31311	139366	3205980
WM Size1	10 (Max.13)	29 (Max.40)	113 (Max.167)	85 (Max.111)	123 (Max. 279)
CS Size1	4 (Max.6)	2 (Max. 5)	43 (Max.120)	13 (Max. 73)	138 (Max. 881)
Token Size1	15 (Max.19)	43 (Max.67)	512 (Max.863)	187 (Max.586)	2968 (Max.8317)

표 1의 첫번째 행에서 규칙의 수는 프로그램내의 규칙의 갯수를 의미하며, 두번째 행에서 조건의 수는 규칙의 조건부에 존재하는 조건의 수를 의미하며, 괄호안의 수는 공통 조건의 수를 말한다. 세번째 행에서 사이클 수는 해당 예제 프로그램 수행시 소요된 추론 사이클 수를 말한다. 네번째 행에서 전체 시간1은 기존의 방법으로 단일 네트워크를 이용하여 프로그램을 실행할 경우의 수행 시간이며, 다섯번째 행에서 매칭 시간1은 전체 시간 1중에서 매칭에서만 소비된 시간이다. 이때, 프로그램의 수행 특성을 보여줄 수 있는 요소로서 평균 작업 메모리의 크기(WM Size1), 충돌 집합의 평균 크기(CS Size1), 그리고 네트워크 내에 저장된 토큰의 평균 갯수(Token Size1)를 각각 6행에서 8행에 표시하였다. 괄호 안의 크기는 각각의 최대 크기를 말한다.

본 논문에서 제안한 분할 알고리즘을 적용하기 위해서는 해당 OPS5 프로그램 내에서 규칙들 사이의 공통조건에 관한 최대수를 얻어야 한다. 각 예제 프로그램을 분석하여 얻은 규칙들 사이의 최대 공통조건수를 표 2에 나타내었다. 표 1과 2를 참조하면 MAB의 경우 전체 규칙수가 20개인데 이중에서 최대 공통조건수

가 3개이다. 그리고 시뮬레이션에 사용된 나머지 예제 프로그램인 Nqueen,Mincut,Toru 프로그램에서 최대 공통조건수는 3개이며, Tourney 경우에서만 11개이다. 이때 얻어진 최대 공통조건수는 분할 알고리즘을 적용하기 위한 초기 값으로 사용하였다.

표 2. OPS5 예제 프로그램의 최대 공통조건수
Table 2. The maximum number of common conditions in OPS5 example programs.

	M A B	Nqueen	Mincut	Toru	Tourney
최대 공통조건수	3	3	3	3	11

표 3. OPS5 예제 프로그램에서 분할된 그룹수
Table 3. The number of partitioned groups in OPS5 example programs.

	MAB	Nqueen	Toru	Mincut	Tourney
분할 그룹수	6	4	8	4	3

표 4. 병렬 매칭 시간
Table 4. Execution time of parallel matching.

프로그램	M A B	Nqueen	Toru	Mincut	Tourney
전체 시간2	1214	3369	38488	218777	3260006
NET A 시간	282 (40%)	1358 (64%)	15907 (51%)	98457 (49%)	2847266 (89%)
NET B 시간	429 (61%)	1247 (59%)	17029 (54%)	107679 (54%)	361534 (11%)
WM Size2	10 (Max.13)	29 (Max.40)	113 (Max.167)	85 (Max.111)	123 (Max.279)
CS Size2	4 (Max. 6)	2 (Max. 5)	43 (Max.120)	13 (Max. 73)	138 (Max.881)
Token Size2	16 (Max.21)	55 (Max.100)	672 (MaX.1135)	261 (Max.674)	4012 (Max.9827)
병렬수행시 전체수행시간	932	2122	22581	120320	2898472
병렬수행시 매칭시간	429	1358	17029	107669	2847266
매칭단축비율	39.2%	35.8%	45.6%	46.0%	11.1%

표 3은 본 논문에서 제안된 규칙 분할 알고리즘을 각각의 프로그램에 적용하여 얻어진 분할 그룹 수이다. 이때, 분할된 그룹의 수는 3~8개로 구분됨을 확인하였다.

표 4는 분할된 그룹들을 2개의 프로세서에 할당하여 병렬 매칭 방법을 적용한 경우의 각 프로그램에 대한 시뮬레이션 수행결과를 표시한다. 표 4의 첫번째 행에서 전체 시간2는 제안된 방법으로 프로그램을 수행할 경우에 소비된 시간이다. 이 시간은 네트워크를 2개(A와 B)로 구분하였을 경우에, 네트워크 A에서 매칭을 수행한 후에 네트워크 B에서 매칭을 수행하였을 경우의 수행 시간을 표시한 것이다. 그러므로, 실제 병렬 환경에서의 수행시간은 (전체 시간 2 - MIN { NET A시간,NET B시간 })이다. 두번째 행에서 NET A 시간은 네트워크 A에서 매칭을 수행하는데 소비된 시간이고, 세번째 행에서 NET B 시간은 네트워크 B에서 매칭을 수행하는데 소비된 시간이다. 두번째와 세번째 행에서 괄호 안의 비율은 표 1의 다섯번째 행의 매칭 시간1을 100으로 놓았을 경우의 상대적인 비율이다. 이때, 작업 메모리의 평균 크기(WM Size2), 충돌 집합의 평균 크기(CS Size2), 그리고 네트워크 내에 저장된 토큰의 평균 갯수(Token Size2)를 각각 4행에서 6행에 표시하였다. 7행에서 8행까지는 프로세서가 2개인 병렬 환경인 경우에 전체 수행 시간과 매칭에 소비된 시간을 표시한다.

예를 들어 20개의 규칙을 가진 MAB(Monkey and Banana) 프로그램에 분할 알고리즘을 적용하면, 단계 2를 적용하였을 경우에 3개의 규칙이 공통 조건을 가진 경우에 해당되어 Mb0(C1,C2), Mb1(C1,C2,C3)이 속한 그룹을 Mb2(C1,C2,C3,C4)가 속한 그룹에 병합한다. 단계 3의 첫번째 수행에서 공통조건수 k가 3인 경우 공통조건 C1,C2,C3에 대한 그룹 G1은 단계 2의 규칙을 고려하여 규칙 Mb0(C1,C2), Mb1(C1,C2,C3), Mb2(C1,C2,C3,C4), Mb3(C1,C2,C3,C5)를 포함한다. 단계 3의 두번째 수행에서 공통조건수 k는 2가 되어 9개의 규칙이 그룹 G_2, G_3, G_4에 속하며, 단계 3의 세번째 수행에서 공통조건수 k는 1이 되어 3개의 규칙이 그룹 G_5에 속한다. 단계 3의 수행으로 4개의 규칙이 나머지 그룹 G_6에 속하게 된다. 6개의 그룹으로 분할된 MAB에서 각 그룹에 대한 변수의 갯수는 G_1 4, G_2 5, G_3 7, G_4 5, G_5 7, G_6 4개이다. 할당 알고리즘을 적용하여 단계 1에서 그룹 A(G_1, G_2, G_4)와 그룹 B(G_3, G5)로 나누었으며, 변수의 갯수를 균등하게 하기 위하여 나머지 그룹 G_6에서 변수를 포함하는 2개의 규칙을 각 그룹에 할당하였다. 이때, 각각 하나 규칙을 각 그룹 A와 B에 할당하여 변수의 갯수를 두 그룹 모두 16개로 하였다. 나머지 그룹 G_6에 남아 있는 규칙은 변수를 포함하고 있지 않은 규칙으로 다음 단계조건을 적용하여 규칙을 그룹 A와 B에 할당한다. 위의 표들에서 측정한 시간단위는 SUN4/SLC 워크스테이션에서 OPS5 인터프리터를 수행할 경우 LISP의 내장함수인 get- internal-run-time 함수를 이용하여 측정한 사이클 수를 의미하며 단위는 10000(만)단위로 하였다.

3. 성능 분석 및 토론

OPS5 예제 프로그램을 수행할 경우에 있어서, 2개의 프로세서를 가진 병렬 환경에서의 병렬 매칭 수행 시간 결과는 표4와 같다. 이때, 병렬 매칭 환경에서 네트워크를 2개(네트워크 A와 네트워크 B)로 구분하였을 경우에 각 네트워크에서의 매칭 수행 시간을 기존의 단일 매칭 환경에서의 매칭 수행시간을 기준으로 한 상대적인 시간 비율로 환산하여 수행시간을 비교하였다.

그림 5는 예제 프로그램의 매칭 시간을 비교하기 하기 위한 그래프이다. 그림 5에서 첫번째 막대 그래프(A)는 기존의 방법으로 매칭을 수행할 경우에 각 프로그램에서 차지하는 매칭 소요시간(표 1의 매칭 시간1)이다. 이는 이후의 수행시간을 환산하여 표시하기 위한 기준으로 사용하였다. 두번째 막대 그래프(B = C + D)는 본 논문의 방법으로 수행시 네트워크 A에서의 매칭 소요시간(표 4의 NET A시간)과 네트워크 B에서의 매칭 소요시간(표 4의 NET B시간)을 더하여 첫번째 막대 그래프에 대한 상대적인 비율로 나타낸 것이다. 세번째와 네번째 막대 그래프는 각각 네트워크 A에서 매칭 소요시간(C), 네트워크 B에서 매칭 소요시간(D)을 첫번째 막대 그래프(A)의 상대적인 시간 비율로 환산하여 표시한 것이다. 표 4와 그림 5의 그래프를 통하여 2개의 프로세서를 사용하여 병렬로 매칭을 수행하였을 경우에 매칭시간의 단축된 비율((MAX(C,D) / A) * 100)을 확인할 수 있었다. MAB의 경우 39%, Nqueen의 경우 36%, Toru의 경우 46%, Mincut의 경우 46%, Tourney의 경우 11%로 매칭 시간이 각각 단축되었다. 그림 5를 살펴보면 Nqueen의 경우 단일 프로세서 환경에서 기존의 네트워크를 2개로 분리하여 네트워크 A와 네트워크 B에서 순차적으로 매칭을 수행한 시간(B=C+D)이 네트워크를 분리하지 않은 경우의 매칭시간(A)에 비해 123%로 걸린 것으로 나타났다. 이는 네트워크를 분리한 후의 오버헤드가 다른 예제 프로그램과 비교할 때 상대적으로 높기 때문에 발생한 것으로 보여지며, Tourney 프로그램의 경우 다른 예제 프로그램들보다 상대적으로 매칭시간 단축율이 낮은 이유는 한 규칙이 반복적으로 많이 사용되어 전체 수행시간의 47%를 차지하기 때문에 발생한 것으로 보여진다.

제안된 규칙 분할 및 할당 알고리즘은 프로그램을 로드할때 규칙을 분할하고 할당하는 정적 알고리즘이다. 이는 프로그램을 로드하여 네트워크를 만드는 과정에 이루어지므로 기존의 방법에 비해 오버헤드가 크지 않다. 그리고, 공통 조건에 따라 그룹을 분할하고 할당함으로써 선택된 OPS5 프로그램으로부터 생성된 그룹의 크기가 3~8개로 크지 않음으로써 네트워크 분할로 인하여 여분의 메모리를 사용하는 것 이외에 커다란 오버헤드는 존재하지 않는다. 표 4에 나타난바와 같이 본 논문에서 제안된 방법이 기존의 방법보다 토큰의 수가 약 1.28배 크게 나타났는데 이는 네트워크를 2개로 분리함으로써 네트워크에 저장된 토큰수가 증가됨으로써 여분으로 사용된 메모리의 양을 의미한다.

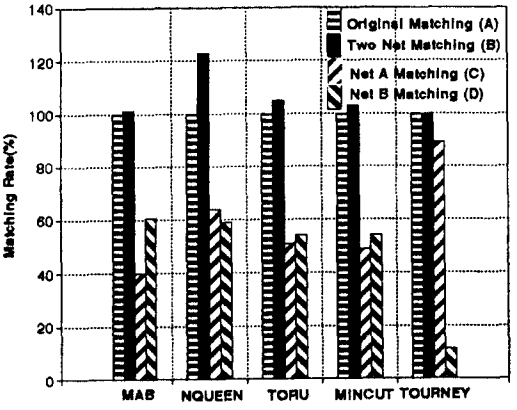


그림 5. 매칭시간 비교
Fig. 5. Comparison of matching time.

V. 결 론

본 논문에서는 생성 시스템의 효율성을 증대시키기 위하여 전체 수행과정 중에서 대부분의 수행시간을 차지하는 매칭과정을 다중 프로세서로 처리함으로써 전체 수행시간을 단축하기 위한 병렬 매칭 알고리즘을 제안하였다. 이를 위해 기존의 단일 프로세서 환경을 기반으로 만들어진 RETE 네트워크[12]가 가지고 있는 병렬성을 최대한 활용하기 위하여, 단일 프로세서환경에서 네트워크를 공유하던 것을 다중 프로세서 환경에서도 그대로 유지하도록 하여 다중 프로세서 환경에서 불필요한 계산과정을 줄이도록 하였다. 이를 위한 구체적인 접근방법으로 규칙레벨의 병렬성을 활용하여, 규칙들의 조건부사이의 유사성을 고려하여 규칙들을 분할하여 하나의 프로그램을 여러 개의 그룹으로 나누고, 분할된 규칙의 그룹을 매칭 소요시간에 비중이 큰 것을 중심으로 고려하여 프로세서에 균등히 할당하기 위한 새로운 병렬 매칭 알고리즘을 제안하였다. 이를 수행하는 과정에서 병렬환경의 구축을 위해 네트워크를 분할함으로써 발생하는 오버헤드는 크게 나타나지 않았으며(1%~13%), 컴파일 타임에 규칙 레벨의 병렬성은 한 규칙이 반복적으로 많이 사용되는 프로그램에서는 성능향상비율(11%)이 낮았지만 대부분의 프로그램에서는 성능향상비율(36~46%)이 높게 나타났다.

앞으로는 매칭과정에 한정하여 수행하였던 접근방법에서 탈피하여 3단계 추론 사이클 전체에 관한 효율적인 병렬환경구축으로 생성 시스템의 효율성을 극대화할 수 있는 방향으로 연구가 계속되어야 할 것이다.

참 고 문 헌

[1] L.Brownston, R.Farrell, E.Kants, N.Martin, Programming Expert Systems in OPS5, Addison-Wesley, pp.19-64, 1985.

[2] Charles L. Forgy, "Rete : A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence 19, pp.17-37, 1982.

[3] Michael A. Kelly, Rudolph E. Seviora, "A Multiprocessor Architecture for Production System Matching", In Proceedings of AAAI-87, pp.36-41, Feb. 1987.

[4] Anoop Gupta, Charles Forgy, Allen Newell, Robert Wedig, "Parallel Algorithm and Architectures for Rule-Based Systems", The 13rd Annual International Symposium on Computer Architecture, pp.28-37, 1986.

[5] Toru Ishida, "Optimizing Rules in Production System Programs" In Prodings of AAAI-88, pp.699-704, Aug. 21-26, 1988.

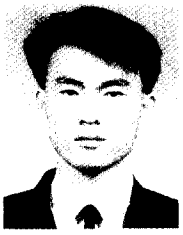
[6] A. O. Oshisanwo P.P. Dasiewicz, "A Parallel Model and Architecture for Production Systems", Proceedings of the International Conference on Parallel Processing, pp.147-153, 1987.

[7] M.F.M Ternorio, D. I. Moldovan, "Mapping Production systems into Multiprocessors", Proceedings of the International Conference on Parallel Processing, pp.466-472, Aug. 1985.

[8] Shigeru Oyanagi, Sumikazu Fujita, etc, "A Highly Parallel Execution Model for Production Systems", Pacific Rim International Conference on Artificial Intelligence '90, pp.781-786, 1990.

[9] Toru Ishida "Parallel Rule Firing in Production Systems", IEEE Trans. on Knowledge and Data Engineering, Vol.3, No.1, pp.11-17, March 1991.

저 자 소 개



姜 勝 一(正會員)

1985년 3월~1992년 2월 숭실대학교 전자공학과 (공학사). 1992년 3월~1994년 2월 숭실대학교 대학원 전자공학과 (공학석사). 1994년 3월~현재 대우전자 SYSTEM 사업부 SYSTEM 개발팀 연구원. 관심분야는 인공지능,전문가 시스템, Intelligent Building System(IBS) 등임.

鄭 圭 植(正會員)

1975년 3월~1979년 2월 서울대학교 전자공학과 (공학사). 1979년 3월~1981년 2월 한국과학기술원 전산학과 (이학석사). 1981년 2월~1984년 7월 금성사 중앙연구소 선임연구원. 1984년 9월~1990년 8월 University of Southern California 컴퓨터공학(공학석사, 공학박사). 1993년 12월~1994년 3월 IBM Watson 연구소 방문연구원. 1990년 9월~현재 숭실대학교 전자공학과 부교수. 관심분야는 인공지능,패턴인식,컴퓨터비젼 등임.

尹 鍾 旻(正會員)

1987년 3월~1991년 2월 숭실대학교 전자공학과 (공학사). 1991년 3월~1993년 2월 숭실대학교 대학원 전자공학과 (공학석사). 1993년 3월~현재 숭실대학교 대학원 전자공학과 박사과정 재학중. 관심분야는 인공지능, 패턴인식, 컴퓨터비젼, 문자인식, 신경망 등임.