행렬을 이용한 FMS에서의 교착상태 탐지 및 회피 알고리즘에 대한 연구

The Study on the Deadlock Detection and Avoidance Algorithm Using Matrix in FMS

송 유 진*, 이 종 근
(Yu-Jin Song and Jong-Kun Lee)

Abstract : The modern production systems are required to produce many items. This is due to the fact that society has become more complex and the customers' demands have become more varied. The demand for complex production systems of various purposes, which can flexibly change the content of work, has increased. One of such production systems is FMS (Flexible Manufacturing System). Limited resources must be used in FMS when a number of working procedures are simultaneously being undertaken because the conditions of stand-by job processes cannot be changed. Researchers are currently being conducted to determine ways of preventing deadlocks. In this study, we proposes the algorithm for detection and recovery of a deadlock status using the DDAPN(Deadlock Detection Avoidance Petri Net). Also, we apply the proposed algorithm has a feature to the FMS.

Keywords : DDAPN, FMS, deadlock, detection, avoidance, matrix, petri net, resource share place, algorithm

I. 서론

초기의 생산 시스템은 단일 품종을 대량으로 생산하는데 유리하도록 설계되었다. 그러나 사회가 복잡해지고 소비자의 취향이 다양화함에 따라 다양한 품종을 생산할 수 있는 생산 시스템을 요구하고 있다. 따라서 고정된 방식의 생산시스템보다는 작업 내용을 쉽게 변경할 수 있는 유연성을 가진 생산 시스템의 필요성이 증가하고 있다.

작업 내용의 변경이 용이한 생산시스템을 유연생산시스템(FMS : Flexible Manufacturing System)이라고 한다. 유연생산시스템에서는 일반적으로 여러 대의 기계와 여러 대의 이동 로봇을 사용하여 유연성을 높인다. 그러나 이런 자원은 한정된 양만을 이용할 수 있으므로 따라서 자원과 자원들의 연결 방법의 선택에 따라 생산의 효율이 크게 달라진다. 이렇게 한정된 자원들로부터 최대의 효율을 내기 위해서 어떻게 자원들을 연결할 것인가가 스케쥴링이고, 또한 이러한 효율성 높은 스케쥴링의 연구와 함께 중요한 부분을 차지하고 있는 연구분야가 바로 교착상태 예방에 관한 연구이다.

교착상태(deadlock)란 다중 태스크 처리에 있어서 상대방이 점유하고 있는 자원을 서로 요구하는 처리가 동시에 발생하게 되면 어느 자원도 해제되지 못하여 처리가 진행되지 않거나 지연되는 상태를 말한다. 유연생산시스템에서는 여러 개의 작업 공정들을 수행하면서 제한된 수의 자원을 사용하여야 한다. 따라서 대기중인 작업 프로세스가 자신의 상태를 바꾸지 못하는 교착상태가 일어날 수 있다. 따라서 현재 이러한 교착상태의 예방을 위한 연구가 활발히 진행되고 있다[1-12]. 교착상태 예방은 다시 교착상태 방지(prevention)와 교착상태 회피(avoidance)로 나눌 수 있다. 방지는 유연생산시스템의 설계 단계에서부터 교착상태가 절대 일어날 수 없도록 설계하는 것이고, 회피는 계속 시스템의 상태를 감시하면서 교착상태가 발생하지 않도록 유도하는 방법이다. 한편, 회복(recovery)은 시스템에서 교착상태가 발생하는지 아닌지를 감시만하고 만약 교착상태가 발생하면 회복시켜주는 방법이다.

본 연구에서는 유연생산시스템의 상태를 본 연구에서 정의하고 있는 DDAPN(Deadlock Detection Avoidance Petri Net)과 행렬(matrix)이론을 이용하여 시스템의 교착상태를 탐지하여 이를 회피하는 알고리즘을 제안하고자 한다.

이 연구의 구성은 다음과 같다.

먼저 2장에서는 유연생산시스템과 교착상태에 관한 연구를 위해 새로운 패트리넷 DDAPN을 정의하고, 3장에서 본 연구의 알고리즘을 위해 필요한 행렬에 대해 정의한다. 4장에서는 교착상태의 탐지와 회피 알고리즘을 예제를 들어 제안하고, 5장에서는 4장에서 제안된 알고리즘을 이용하여 여러 사례들에 적용한다. 그리고 6장에서 앞으로의 연구방향과 함께 결론을 맺는다.

II. FMS 와 DDAPN

1. 유연생산시스템(FMS)

유연생산시스템을 모델링하는 문제는 시스템의 특성상 연속적인 시스템보다는 이산 사건 시스템으로 잘 표현될 수 있다. 이러한 이산사건 시스템을 모델링하고 분석하는 도구로는 오토마타, 패트리넷 등이 있는데 공유된 자원, 동시 발생 등을 간결하게 표현할 수 있어서 유연생산시스템을 모델링하는데 매우 적합하다.

또한, 기계, 이동 로봇, 부품과 같은 자원을 이용해 다양한 품종을 생산할 수 있는 생산 시스템인 유연생산시스템은 작업을 진행하는 복수개의 과정과 이러한 작업 프로세스들이 원활히 수행되도록 자원들을 분배해주는 자원의 수요 공급을 담당하는 부분들로 구성되어 있다. 따라서 자원이 많으면 많을수록 생산의 효율이 높아질 것이다. 그러나 이런 자원은

* 책임저자(Corresponding Author)
논문접수 : 2004. 7. 20., 채택확정 : 2004. 11. 3.
송유진, 이종근 : 창원대학교 컴퓨터공학과
(syj@changwon.ac.kr/jklee@sarim.changwon.ac.kr
※ 본 연구는 과학 기술부 목적기초연구(R08-2004-000-10029-0) 지원으로 수행되었음.

한정된 양만을 이용할 수 있고, 또한 한번에 하나의 작업을 하며 프로세스가 진행된다. 그러므로, 사용가능한 자원은 복수개이나 작업 단계에 있어서 한 순간 필요한 자원은 하나이다. 그래서 이러한 유연생산시스템을 패트리넷으로 모델링 함에 있어서도 작업의 단계마다에는 하나의 자원이 들어가 하나의 자원이 출력되어 진행된다.

본 연구에서는 이러한 특징을 나타내고 있는 FMS를 모델링 하기 위해 DDAPN을 정의하여 모델링에 이용하고자 한다.

2. DDAPN

(Def 2.1) DDAPN

$DDAPN = (P, R, T, I, O, M_0)$ 여기서,

$P = \{p_1, p_2, \ldots\ldots, p_n\}$:작업플레이스들의 집합 $(n \geq 0)$

$R = \{r_1, r_2, \ldots\ldots, r_m\}$:자원공유플레이스들의 집합 $(m \geq 0)$

$T = \{t_1, t_2, \ldots\ldots\ldots, t_k\}$:트랜지션들의 집합 $(k \geq 0)$

$P \cap R = \phi \quad P \cap T = \phi \quad R \cap T = \phi$

$\forall p \in P \qquad |\bullet p| = 1 \quad and \quad |p \bullet| = 1$

$I : P \times T \rightarrow N \quad R \times T \rightarrow N$, I 는 입력함수

$O : T \times P \rightarrow N \quad T \times R \rightarrow N$, O는 출력함수

$P \neq 0 \quad and \quad R \neq 0 \quad and \quad T \neq 0$

$M_0 \in M = \{M | M : P \rightarrow N\}$, M_0는 초기마킹

N : 정수들의 집합

여기서, P는 유연생산시스템의 패트리넷 모델에서 유연생산시스템의 작업을 나타내기 위한 플레이스들의 집합을 말하고, R은 작업을 위해 필요한 자원들의 상태를 표현하는 자원 공유 플레이스들의 집합을 나타낸다. 또한 T는 다른 작업을 위한 조건 여부를 체크하는 트랜지션들의 집합이고, I는 작업 플레이스나 자원공유 플레이스들에서 트랜지션으로 입력되는 입력함수, O는 트랜지션에서 작업 플레이스나 자원공유 플레이스로 출력되는 출력함수를 말한다.

이를 예를 들면 다음 그림1과 같다.

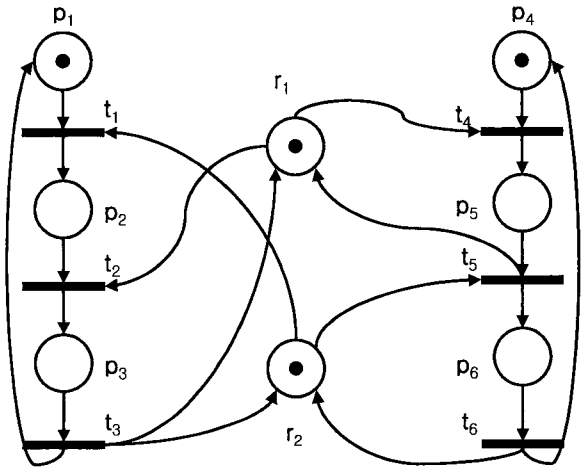

그림 1. DDAPN 예제.

Fig. 1. DDAPN example.

그림 1에서 작업 플레이스들의 집합은 $P = \{p_1, p_2, p_3, p_4, p_5, p_6\}$ 이고, 자원공유 플레이스들의 집합은 $R = \{r_1, r_2\}$ 이다. 트랜지션들의 집합은 $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ 이다.

III. 행렬

이 장에서는, 교착상태의 탐지와 회피 알고리즘에 대한 연구를 위해 필요한 행렬을 정의하고자 한다. 즉, DDAPN 모델을 행렬로 표현하고 이를 이용해 교착상태를 탐지하고 회피하고자 한다.

먼저 DDAPN을 행렬 M_{PR}로 표현하는 방법을 정의한다.

정의 1 : M_{PR} 은 다음을 만족하는 정방행렬이다.

$$M_{PR} = \begin{array}{c} \begin{matrix} p_1 & p_2 & \cdots & \cdots & p_n & r_1 & \cdots & \cdots & r_m \end{matrix} \\ \begin{bmatrix} p_1p_1 & \cdots & \cdots & \cdots & p_1p_n & p_1r_1 & \cdots & p_1r_m \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ p_np_1 & \cdots & \cdots & \cdots & p_np_n & p_nr_1 & \cdots & p_nr_m \\ r_1p_1 & \cdots & \cdots & \cdots & r_1p_n & r_1r_1 & \cdots & r_1r_m \\ \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots \\ r_mp_1 & \cdots & \cdots & \cdots & r_mp_n & r_mr_1 & \cdots & r_mr_m \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ \cdots\cdots \\ \cdots\cdots \\ p_n \\ r_1 \\ \cdots\cdots \\ r_m \end{matrix}$$

여기서,

$$M_{PR} = \begin{cases} p_ip_j & (1 \leq i, j \leq n) \\ r_{i-n}p_j & \begin{pmatrix} n+1 \leq i \leq n+m \\ 1 \leq j \leq n \end{pmatrix} \\ p_ir_{j-n} & \begin{pmatrix} 1 \leq i \leq n \\ n+1 \leq j \leq n+m \end{pmatrix} \\ r_{i-n}r_{j-n} & (n+1 \leq i, j \leq n+m) \end{cases}$$

<p_ip_j의 값들 중에서>

1 : 각 작업 플레이스간의 트랜지션을 통한 직접적인 연결성이 있다.

0 : 각 작업 플레이스간의 트랜지션을 통한 직접적인 연결성이 없다.

<$r_{i-n}p_j$의 값들 중에서>

1 : 프로세스 진행을 위해 자원공유 플레이스에 있는 자원이 필요하다.

0 : 프로세스 진행을 위해 자원공유 플레이스에 있는 자원이 필요 없다.

<p_ir_{j-n}의 값들 중에서>

1 : 프로세스 진행을 위해 트랜지션 점화 후 자원공유 플레이스에 자원을 돌려준다.

0 : 트랜지션 점화 후 자원공유 플레이스에 자원을 돌려주지 않는다.

<$r_{i-n}r_{j-n}$의 값들 중에서>

1 : 자원공유 플레이스간의 관계는 프로세스 진행과 관계가 없으므로 1의 값은 없다.

0 : 자원공유 플레이스간에는 프로세스 진행과 관계가 없으므로 모든 값이 0이다.

정의 1에 의해 그림 1의 모델을 M_{PR}로 나타내면 다음과 같다.

$$M_{PR}=\begin{array}{c} \begin{matrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 & r_1 & r_2 \end{matrix} \\ \begin{bmatrix} 0 & 1_{j1} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1_{j1} & 0 & 0 & 0 & 0 & 0 \\ 1_{j1} & 0 & 0 & 0 & 0 & 0 & 1_{j1} & 1_{j1} \\ 0 & 0 & 0 & 0 & 1_{j2} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1_{j2} & 1_{j2} & 0 \\ 0 & 0 & 0 & 1_{j2} & 0 & 0 & 0 & 1_{j2} \\ 0 & 0 & 1_{j1} & 0 & 1_{j2} & 0 & 0 & 0 \\ 0 & 1_{j1} & 0 & 0 & 0 & 1_{j2} & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \\ r_1 \\ r_2 \end{matrix}$$

여기서, j1은 패트리넷 모델에서 JOB1을 의미하고, j2는 JOB2를 의미한다. 또한, 그림 1에서 작업 시작 플레이스는 p1과 p4이다. 결국, p1이 p2로 가기 위해서는 r2에서 자원을 받아야 수행 할 수 있고, p4에서 p5로의 작업 진행을 위해서는 r1에서 자원을 받아야 수행 가능하다. 그리고 p2에서 p3으로의 작업 수행을 위한 점화 후에는 r1과 r2에 각각 자원을 돌려준다. 이러한 과정이 행렬의 표현으로 가능하며 이를 자세히 설명하면 다음 그림 2와 같다.

정의 2 : M_P 는 다음을 만족하는 정방행렬이다.

$$M_P=\begin{array}{c} \begin{matrix} p_1 & p_2 & \cdots\cdots & \cdots\cdots & p_n \end{matrix} \\ \begin{bmatrix} p_1p_1 & \cdots\cdots & \cdots\cdots & \cdots\cdots & p_1p_n \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots & \cdots\cdots \\ p_np_1 & \cdots\cdots & \cdots\cdots & \cdots\cdots & p_np_n \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ \cdots\cdots \\ \cdots\cdots \\ p_n \end{matrix}$$

여기서,

$$M_P=\{p_ip_j \quad (1\le i,j\le n)$$

행렬 M_{PR}에서 작업 플레이스 부분만을 나타내는 행렬 M_P는 정의 2에 의해 다음과 같이 나타낸다.

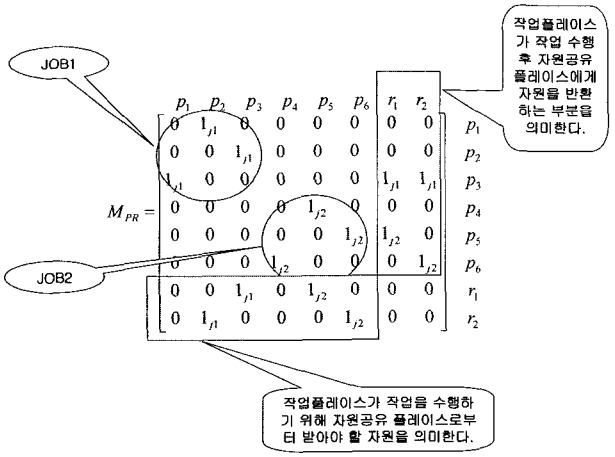

그림 2. 그림 1의 행렬 M_{PR}.
Fig. 2. Matrix M_{PR} of the fig. 1.

$$M_P=\begin{array}{c} \begin{matrix} p_1 & p_2 & p_3 & p_4 & p_5 & p_6 \end{matrix} \\ \begin{bmatrix} 0 & 1_{j1} & 0 & 0 & 0 & 0 \\ 0 & 0 & 1_{j1} & 0 & 0 & 0 \\ 1_{j1} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1_{j2} & 0 \\ 0 & 0 & 0 & 0 & 0 & 1_{j2} \\ 0 & 0 & 0 & 1_{j2} & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ p_5 \\ p_6 \end{matrix}$$

정의 3 : Cycle

패트리넷 모델에서 한 프로세스의 시작 작업 플레이스와 마지막 작업 플레이스의 바로 이전 플레이스가 같다면 이는 사이클을 형성하고 있음을 나타낸다.

즉, 행렬 M_P에서 하삼각행렬에 1의 값을 가지게 되면 사이클이 된다.

왜냐하면, 상삼각행렬은 프로세스가 순방향으로 진행하는 부분을 나타내고 하삼각행렬은 역방향의 진행을 표현하므로 순방향에서 역방향으로 진환하는 부분이 존재하는 것이 사이클이기 때문이다.

따라서 다음과 같은 조건을 만족하는 p_ip_j가 있을 때 사이클이 존재한다.

$$p_ip_j\begin{bmatrix} p_ip_j=1 \\ i>j \end{bmatrix}$$

그림 3은 그림 1의 DDAPN 예제에 정의 3을 적용하여 이를 만족하는 p_ip_j 들을 나타내고 있다.

그림 3에서 JOB1의 작업 플레이스와 JOB2의 작업 플레이스들이 서로 겹치지 않고, 시작 플레이스가 JOB1과 JOB2에 각각 있으므로 이는 JOB1과 JOB2가 동시 수행이 가능하다고 볼 수 있다. 따라서 교착상태를 고려할 때 JOB1과 JOB2의 동시수행도 같이 고려해야 하므로 행렬을 다음과 같이 M_J로 정의한다.

$$M_J=\begin{array}{c} \begin{matrix} p_1,p_4 & p_2,p_5 & p_3,p_6 \end{matrix} \\ \begin{bmatrix} 0 & 1_{j1},1_{j2} & 0 \\ 0 & 0 & 1_{j1},1_{j2} \\ 1_{j1},1_{j2} & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} p_1,p_4 \\ p_2,p_5 \\ p_3,p_6 \end{matrix}$$

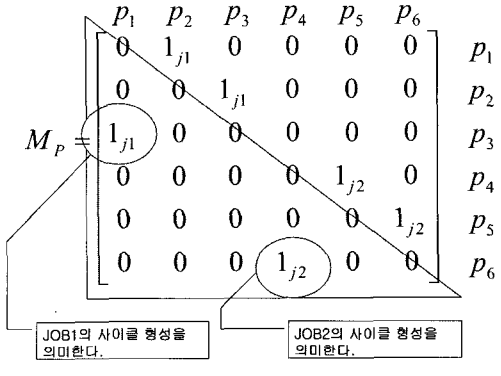

그림 3. 그림 1의 작업 플레이스 부분의 행렬 M_P.
Fig. 3. Matrix M_P of the job place region of the fig. 1.

M_J의 상태에 맞추어 자원공유 플레이스들의 상태를 같이 고려하여야 교착상태의 탐지 및 회피에 관한 해석이 나오게 되므로 위의 M_J 행렬에 맞추어 자원공유 플레이스의 값도 같이 고려하여 최종 행렬을 구성한다.

이를 나타내면 다음의 행렬 M_{JR} 이 된다.

$$M_{JR}=\begin{array}{c} \begin{matrix} p_1,p_4 & p_2,p_5 & p_3,p_6 & r_1 & r_2 \end{matrix} \\ \begin{bmatrix} 0 & 1_{j1},1_{j2} & 0 & 0 & 0 \\ 0 & 0 & 1_{j1},1_{j2} & 1_{j2} & 0 \\ 1_{j1},1_{j2} & 0 & 0 & 1_{j1} & 1_{j1},1_{j2} \\ 0 & 1_{j2} & 1_{j1} & 0 & 0 \\ 0 & 1_{j1} & 1_{j2} & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} p_1,p_4 \\ p_2,p_5 \\ p_3,p_6 \\ r_1 \\ r_2 \end{matrix}$$

위의 최종 행렬을 이용하여 이제 교착상태를 탐지하고 회피하는 알고리즘을 제안한다.

IV. 교착상태 탐지 및 회피 알고리즘

1. 교착상태 탐지 과정

이 장에서는 3장에서 제시된 행렬 표현들을 이용하여 모델의 교착상태 여부와 만약 교착상태이면 이를 회피할 수 있는 알고리즘을 제안하고자 한다.

먼저, 행렬 M_{JR}을 이용한 단계별 알고리즘의 현황을 그림 4에 나타내었다.

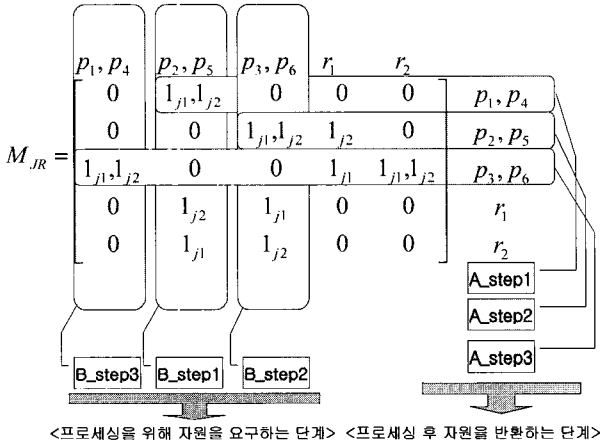

그림 4. 그림 1의 알고리즘을 위한 단계.

Fig. 4. Steps for the algorithm of the fig. 1.

그림 4에서 B_step은 각 단계에 해당되는 작업 플레이스가 수행되기 위해 필요한 트랜지션의 점화전(Before) 자원의 상태이고, A_step은 각 단계에 해당되는 작업 플레이스가 수행되기 위해 필요한 트랜지션의 점화 후(After) 이루어지는 자원의 상태를 의미한다.

또한, 처음 시작 단계는 초기 마킹이 되어 있는 작업 플레이스의 바로 다음에 있는 트랜지션을 중심으로 시작된다.

따라서, 작업 플레이스가 동작하기 위해서 필요한 자원공유 플레이스의 자원들의 상태를 나타내는 테이블이 필요하다. 자원공유 플레이스의 자원들의 상태를 나타내는 테이블은 표 1과 같다.

여기서, 각 단계의 자원상태 값은 다음 정의 4에 의해 구해진다.

정의 4 : 각 단계의 자원 상태 값

B_step i 의 자원의 값

= 전 단계의 자원의 값 - B_step i 의 자원의 값

A_step i 의 자원의 값

= 전 단계의 자원의 값 + A_step i 의 자원의 값

또한, 자원의 숫자 옆의 괄호 안의 내용은 그 자원이 어느 JOB의 진행을 위해 필요로 하는 자원인지를 나타낸다. 즉, 1(j1orj2) 라고 하면 이 자원은 1개가 있는데 이 자원1개는 JOB1에도 사용가능하고, JOB2에도 사용가능함을 나타낸다. 만약, 1(j1)이라고 하면 이 자원 한 개는 반드시 JOB1의 진행을 위해서만 필요한 자원이라는 뜻이다.

표 1에서 B_step2의 r1, r2의 값이 음수가 되면서, 각각 JOB1과 JOB2에 동시에 음수가 된다. 이는 B_step이 작업 플레이스가 수행되기 위해 필요한 트랜지션의 점화전 상태임을 고려할 때, 두 개의 JOB이 동시에 음수가 되므로 필요한 자원 공급이 되지 않아 교착상태에 빠지게 됨을 의미한다. 따라서 이 부분이 교착상태의 원인이 됨을 알 수 있다.

2. 교착상태 회피 과정

이 장에서는 교착상태가 탐지된 후 이를 회피하기 위한 알고리즘을 제안하고자 한다.

그림 1의 DDAPN모델의 교착상태를 회피하는 과정을 설명하면 다음과 같다.

먼저, 교착상태가 일어난 부분의 음수 값을 없애기 위해 필요한 자원을 앞 단계의 A_step에 넣어준다. 즉, B_step2에서 교착상태이므로 앞의 단계인 A_step1의 값을 각각 1(j1orj2)과 1(j1orj2)로 수정한다.

여기서, 앞 단계의 B_step은 수정하지 않는다. 왜냐하면,

표 1. 그림 1의 자원 상태 테이블

Table 1. Resource state table of fig. 1.

Resource Share Place	초기의 자원 상태	B_step1	A_step1	B_step2	A_step2	B_step3	A_step3
r1	1 (j1orj2)	0	0	-1 (j1orj2)	0	0	1 (j1orj2)
r2	1 (j1orj2)	0	0	-1 (j1orj2)	-1 (j1orj2)	-1 (j1orj2)	1 (j1orj2)

표 2. 그림 1의 수정된 자원 상태 테이블(1).

Table 2. Modified resource state table of the fig. 1(1).

Resource Share Place	초기의 자원 상태	B_step1	A_step1	B_step2	A_step2	B_step3	A_step3
r1	1 (j1orj2)	0	1 (j1orj2)	0	1 (j1orj2)	1 (j1orj2)	2 (j1orj2)
r2	1 (j1orj2)	0	1 (j1orj2)	0	0	0	2 (j1orj2)

FMS를 페트리넷으로 모델링 한 것이므로 FMS의 특성상 제품을 만드는데 필요한 기계나 부품을 의미하는 자원을 임의로 수정할 수 없기 때문이다. 따라서, 이와 같이 수정하면 테이블은 표 2와 같이 수정된다.

표 2를 이용해 행렬을 작성하면 다음 그림 5와 같다.

한편, 자원의 상태 테이블에서 마지막은 항상 초기 자원의 상태를 가지고 있어야 계속적인 프로세스 진행이 가능하다.

위의 표 2의 마지막 단계인 A_step3은 이러한 조건을 만족하지 않으므로 이를 수정하여야 한다.

즉, A_step3의 r1에서 1(j1orj2)을 삭제하고, r2에서 1(j1orj2)를 삭제하면 초기의 자원 상태와 같아지므로 boundedness 성질을 만족하며, 계속적인 프로세스 진행이 가능하다. 따라서, 이와 같이 수정하면 테이블은 다음 표 3과 같이 수정된다.

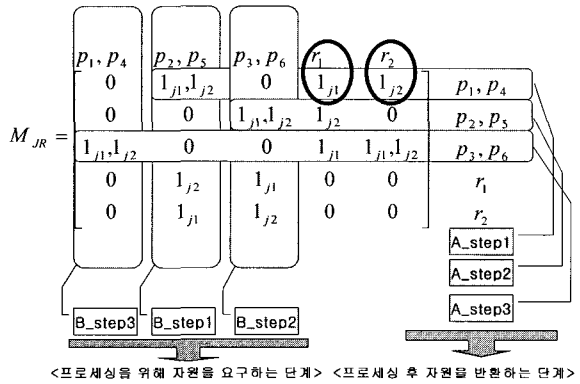

그림 5. 표 2를 적용한 M_{JR}.

Fig. 5. M_{JR} after application of the table 2.

표 3. 그림 1의 수정된 자원 상태 테이블(2).

Table 3. Modified resource state table of the fig 1(2).

Resou_rce Share Place	초기의 자원 상태	B_step1	A_step1	B_step2	A_step2	B_step3	A_step3
r1	1 (j1orj2)	0	1 (j1orj2)	0	1 (j1orj2)	1 (j1orj2)	1 (j1orj2)
r2	1(j1orj2)	0	1 (j1orj2)	0	0	0	1 (j1orj2)

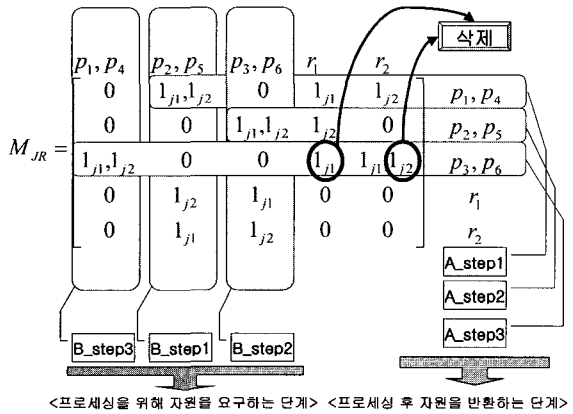

그림 6. 표 3을 적용한 M_{JR}.

Fig. 6. M_{JR} after application of the table 3.

표 3을 이용해 행렬을 작성하면 그림 6과 같다.

3. 교착상태 탐지와 회피 알고리즘

1과 2절에서의 자원 상태 테이블을 이용한 교착상태 탐지와 회피 과정을 알고리즘으로 표현하면 다음과 같다.

```
int main()
{
int n, m, JOB;
n = the number of resource places;
m = the number of B_step + the number of A_step + 1;
JOB = the number of JOB;
int array[n][m];
int j=0;
while ( j <= m-1)
{
    j = Detection(n, m, JOB, array);
    if (j <= m-1)
        Avoidance(j, n, m, array);
}
return 0;
}
int Detection(int n, int m, int JOB, int &array)
{
  int i, j, k;
  for (j=0 ; j< m ; j++)
  {
   int k = 0;
   for(i=0 ; i<n ; i++)
   {
    if(array[i][j] < 0)
       k = | array[i][j] |;
   }
   if (k = JOB)
   {
        cout << "This DDAPN model is DEADLOCK STATUS";
        cout << " in B_step(" <<(j-1)/2+1 << ")";
        return j;
   }
  }
  if (array[0][0] != array[0][m-1] || array[1][0] != array[1][m-1])
  {
     cout << "This DDAPN model is UNBOUNDEDNESS";
      return m-1;
  }
  cout << "This DDAPN model is not Deadlock status";
  return m;
}
int Avoidance(int j, int n, int m, int &array)
{
  int i, k;
  for (i=0; i<n ; i++)
  {
    if(array[i][j-1] < 0)
   array[i][j-2] = array[i][j-2] +1;
  }
  reconstruct the resource state table by (definition 4);
  return 0;
  }
```

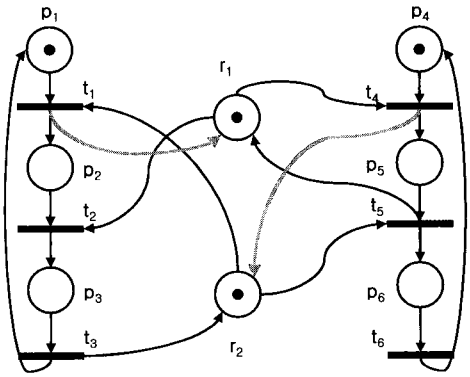

그림 7. 그림 1을 교착상태 회피 알고리즘 적용 한 후의 수정 모델.

Fig. 7. Modification model after application of the deadlock avoidance algorithm in fig. 1.

앞의 교착상태 탐지 및 회피 알고리즘을 적용하여 교착상태가 회피된 최종적인 패트리넷 모델을 그림 6의 행렬을 이용해 작성하면 그림 7과 같다.

그림 7은 이제 그림 1과는 달리 교착상태가 일어나지 않으며, boundedness성질도 만족하고 있음을 알 수 있다.

V. 사례 연구

그림 8은 기계 한 대와 부품 자원을 이용해 제품을 만드는 두 개의 JOB으로 이루어진 FMS 모델이다. 이를 그림 9와 같이 교착상태가 유발되는 DDAPN 모델로 작성하여 이 모델을 이용해 교착상태의 여부를 탐지하고 회피하는 과정을 기술하고자 한다.

그림 9에서 작업 플레이스들의 집합은 $P = \{p_1, p_2, p_3, p_4\}$ 이고, 자원공유 플레이스들의 집합은 $R = \{r_1, r_2\}$ 이다. 트랜지션들의 집합은 $T = \{t_1, t_2, t_3, t_4\}$ 이다.

그림 9의 행렬 표현으로 다음 M_{PR}과 같다.

$$M_{PR} = \begin{array}{c} \begin{matrix} p_1 & p_2 & p_3 & p_4 & r_1 & r_2 \end{matrix} \\ \begin{bmatrix} 0 & 1_{j1} & 0 & 0 & 0 & 0 \\ 1_{j1} & 0 & 0 & 0 & 0 & 1_{j1} \\ 0 & 0 & 0 & 1_{j2} & 0 & 0 \\ 0 & 0 & 1_{j2} & 0 & 1_{j2} & 0 \\ 0 & 1_{j1} & 1_{j2} & 0 & 0 & 0 \\ 1_{j1} & 0 & 0 & 1_{j2} & 0 & 0 \end{bmatrix} \end{array} \begin{matrix} p_1 \\ p_2 \\ p_3 \\ p_4 \\ r_1 \\ r_2 \end{matrix}$$

여기서, j1은 패트리넷 모델에서 JOB1을 의미하고, j2는 JOB2를 의미한다. 또한, 그림 9에서 작업 시작 플레이스는 p1과 p3이다. 결국, p1이 p2로 가기 위해서는 r1에서 자원을 받아야 수행 할 수 있고, p3에서 p4로의 작업 진행을 위해서는 r2에서 자원을 받아야 수행 가능하다. 그리고 p2에서 p1으로의 작업 수행을 위한 점화 후에는 r2에 자원을 돌려주고, p4에서 p3으로의 작업 수행을 위한 점화 후에는 r1에 자원을 돌려준다. 이러한 과정이 행렬의 표현으로 가능하며 이를 자세히 설명하면 그림 10과 같다.

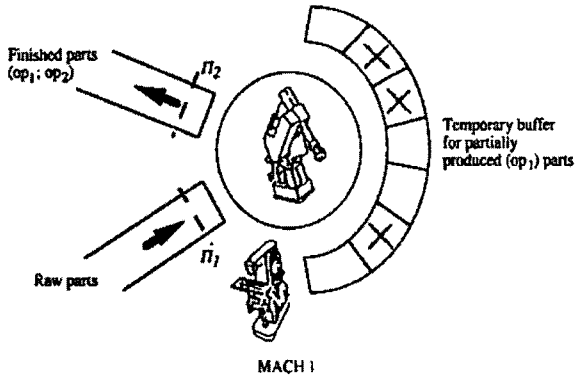

그림 8. 기계 하나, 로봇 하나를 가진 FMS의 구조.

Fig. 8. The schema of a FMS with one machines and one robot.

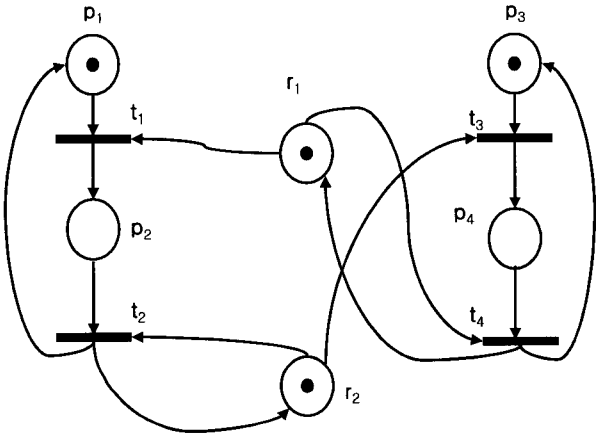

그림 9. 그림 8의 DDAPN.

Fig. 9. DDAPN of the fig. 8.

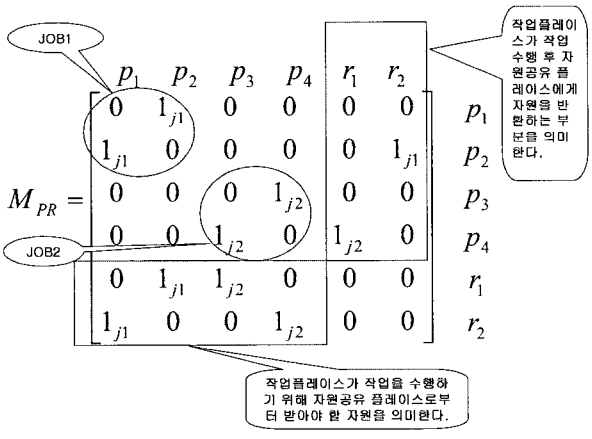

그림 10. 그림 9의 행렬 M_{PR}.

Fig. 10. Matrix M_{PR} of the fig. 9.

행렬 M_{PR}에서 작업 플레이스 부분만을 놓고 볼 때 하삼각 행렬에 1이 되면 이는 정의 3에 의해 사이클을 의미한다. 먼저 M_{PR}의 작업 플레이스 부분만을 나타내는 행렬 M_P는 그림 11과 같다.

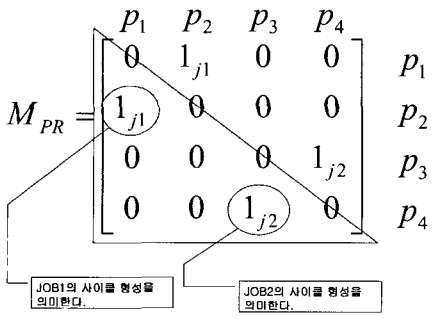

그림 11. 그림 9의 작업 플레이스 부분의 행렬 M_P.
Fig. 11. Matrix M_P of the job place region of the fig. 9.

그림 11 에서 JOB1의 작업 플레이스와 JOB2의 작업 플레이스들이 서로 겹치지 않고, 시작 플레이스가 JOB1과 JOB2에 각각 있으므로 이는 JOB1과 JOB2가 동시 수행이 가능하다고 볼 수 있다. 따라서 교착상태를 고려할 때 JOB1과 JOB2의 동시수행도 같이 고려해야 하므로 행렬을 다음과 같이 M_J 로 정의한다.

$$M_J = \begin{bmatrix} p_1,p_9 & p_2,p_{10} & p_3,p_6 & p_4,p_7 & p_5,p_8 & \\ 0 & 1_{j1},1_{j2} & 0 & 0 & 0 & p_1,p_9 \\ 0 & 0 & 1_{j1},1_{j2} & 0 & 0 & p_2,p_{10} \\ 0 & 0 & 0 & 1_{j1},1_{j2} & 0 & p_3,p_6 \\ 0 & 0 & 0 & 0 & 1_{j1},1_{j2} & p_4,p_7 \\ 1_{j1},1_{j2} & 0 & 0 & 0 & 0 & p_5,p_8 \end{bmatrix}$$

M_J의 상태에 맞추어 자원공유 플레이스들의 상태를 같이 고려하여야 교착상태의 탐지 및 회피에 관한 해석이 나오게 되므로 위의 M_J 행렬에 맞추어 자원공유 플레이스들의 값도 같이 고려하여 최종 행렬을 구성한다. 이를 나타내면 다음의 행렬 M_{JR} 이 된다.

$$M_{JR} = \begin{bmatrix} p_1,p_9 & p_2,p_{10} & p_3,p_6 & p_4,p_7 & p_5,p_8 & r_1 & r_2 & r_3 & \\ 0 & 1_{j1},1_{j2} & 0 & 0 & 0 & 0 & 0 & 0 & p_1,p_9 \\ 0 & 0 & 1_{j1},1_{j2} & 0 & 0 & 1_{j2} & 1_{j1},1_{j2} & 0 & p_2,p_{10} \\ 0 & 0 & 0 & 1_{j1},1_{j2} & 0 & 0 & 0 & 0 & p_3,p_6 \\ 0 & 0 & 0 & 0 & 1_{j1},1_{j2} & 0 & 0 & 0 & p_4,p_7 \\ 1_{j1},1_{j2} & 0 & 0 & 0 & 0 & 0 & 1_{j1},1_{j2} & 1_{j1} & p_5,p_8 \\ 0 & 0 & 0 & 0 & 1_{j1} & 0 & 0 & 0 & r_1 \\ 0 & 1_{j1},1_{j2} & 0 & 0 & 1_{j1},1_{j2} & 0 & 0 & 0 & r_2 \\ 0 & 1_{j2} & 0 & 0 & 0 & 0 & 0 & 0 & r_3 \end{bmatrix}$$

위의 최종 행렬을 이용하여 다음의 알고리즘에 의해 교착상태를 탐지하고 또한 교착상태의 회피가 가능하다.

먼저 행렬 M_{JR}을 이용한 단계별 알고리즘의 현황을 그림 12에 나타내었다.

그림 12에서 B_step은 각 단계에 해당되는 작업 플레이스가 수행되기 위해 필요한 트랜지션의 점화전(before) 자원의 상태이고, A_step은 각 단계에 해당되는 작업 플레이스가 수행되기 위해 필요한 트랜지션의 점화 후(after) 이루어지는 자원의 상태를 의미한다.

또한, 처음 시작 단계는 초기 마킹이 되어 있는 작업 플레이스의 바로 다음에 있는 트랜지션을 중심으로 시작된다.

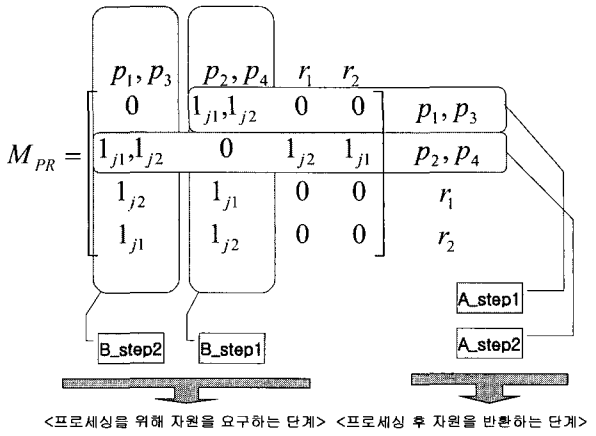

그림 12. 그림 11의 알고리즘을 위한 단계.
Fig. 12. Steps for the algorithm of the fig. 11.

표 4. 그림 9의 자원 상태 테이블.
Table 4. Resource state table of fig. 9.

Resource Share Place	초기의 자원상태	B_step1	A_step1	B_step2	A_step2
r1	1(j1orj2)	0	0	-1(j1orj2)	0
r2	1(j1orj2)	0	0	-1(j1orj2)	0

따라서, 작업 플레이스가 동작하기 위해서 필요한 자원공유 플레이스의 자원들의 상태를 나타내는 테이블이 필요하다. 자원공유 플레이스의 자원들의 상태를 나타내는 테이블은 위와 같다.

표 4에서, r1과 r2는 JOB1과 JOB2의 프로세스 진행을 위해 자원을 내어놓으므로 1(j1orj2)라고 표현한다.

교착상태는 다중 태스크 처리에 있어서 상대방이 점유하고 있는 자원을 서로 요구하는 처리가 동시에 발생하게 되면 어느 자원도 해제되지 못하여 처리가 진행되지 않거나 지연되는 상태를 말하는데, 표 4에서는 B_step2의 r1, r2의 값이 음수가 되면서, 각각 JOB1과 JOB2에 동시에 음수가 된다. 이는 B_step이 작업 플레이스가 수행되기 위해 필요한 트랜지션의 점화전 상태임을 고려할 때, 두 개의 JOB이 동시에 음수가 되므로 필요한 자원 공급이 되지 않아 교착상태에 빠지게 됨을 의미한다. 따라서 이 부분이 교착상태의 원인이 됨을 알 수 있다.

이러한 교착상태를 회피하기 위해서는 먼저, 교착상태가 일어난 부분의 음수 값을 없애야 하는데, 이를 위해 필요한 자원을 앞 단계의 A_step에 넣어준다. 즉, B_step2에서 교착상태이므로 앞의 단계인 A_step1의 값을 각각 1(j1orj2)과 1(j1orj2)로 수정한다.

여기서, 앞 단계의 B_step은 수정하지 않는다. 왜냐하면, FMS를 페트리넷으로 모델링 한 것이므로 FMS의 특성상 제품을 만드는데 필요한 기계나 부품을 의미하는 자원을 임의로 수정할 수 없기 때문이다. 따라서, 이와 같이 수정하면 테이블은 다음 표 5와 같이 수정된다.

표 5의 상태 테이블을 이용하여 행렬을 작성하면 다음 그림 13과 같다.

표 5. 그림 9의 수정된 자원 상태 테이블.
Table 5. Modified resource state table of the fig. 9.

Resource Share Place	초기의 자원상태	B_step1	A_step1	B_step2	A_step2
r1	1(j1orj2)	0	1(j1orj2)	0	1(j1orj2)
r2	1(j1orj2)	0	1(j1orj2)	0	1(j1orj2)

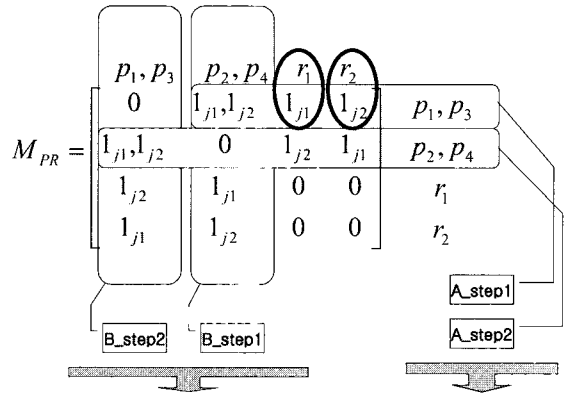

그림 13. 표 5을 적용한 M_{JR}.
Fig. 13. M_{JR} after application of the table 5.

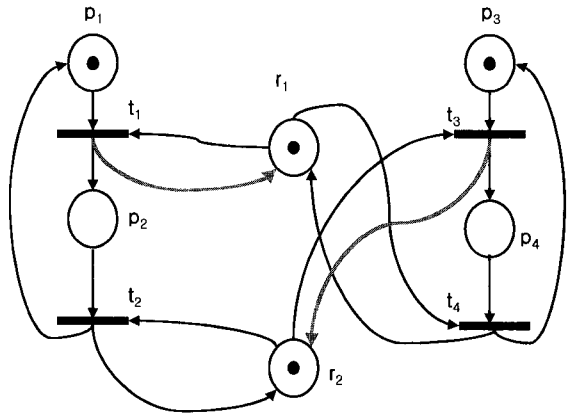

그림 14. 그림 9의 수정모델.
Fig. 14. Modification model of the fig. 9.

또한, 표 5에서, A_step2의 마지막 상태가 초기의 자원 상태와 같으므로 boundedness의 성질도 만족함을 알 수 있다.

따라서, 그림 9의 DDAPN의 교착상태가 회피 되었음을 알 수 있다.

그림 13의 행렬 표현을 페트리넷으로 표현하면 다음 그림 14와 같으며, 이 모델은 교착상태가 일어나지 않는다.

VI. 결론

이 연구에서 우리는 기계, 이동 로봇, 부품과 같은 자원을 이용해 다양한 품종을 생산할 수 있는 생산 시스템인 유연생산시스템에서의 교착상태 탐지와 회피를 위해 행렬과 DDAPN을 새로 정의하여 교착상태 탐지 및 회피 알고리즘을 제안하였다.

유연생산시스템이 작업을 진행하는 복수개의 과정과 이러한 작업 프로세스들이 원활히 수행되도록 자원들을 분배해 주는 자원의 수요 공급을 담당하는 부분들로 구성되어 있고, 또한 이런 자원이 한번에 하나의 작업을 하며 프로세스가 진행되는 특징이 있으므로 이를 작업 플레이스들의 조건이 marked graph인 DDAPN을 새로이 정의하여 알고리즘을 제안하였다. 한편, 이 연구에서 제안한 교착상태 탐지 및 회피 알고리즘을 적용하면 교착상태를 벗어날 수 있는 가능한 모델이 직접 출력됨으로써 수정 모델을 통해 교착상태가 없는 적절한 작업을 수행할 수 있는 장점이 있다.

이제, 앞으로의 연구는 제안된 알고리즘을 다른 교착상태 관련 알고리즘들과 비교 분석하여 그 성능을 측정하고자 한다.

참고문헌

[1] J. C. Corbett "Evaluating deadlock detection methods for concurrent software," *IEEE tr. Software Engineering,* vol. 22(3), 1996.

[2] J. Ezpleta, J. M. Colom And Martinez J., "A petri net based on deadlock prevention policy for flexible manufacturing systems," *IEEE tr. Robotics and Automation,* vol. 11. no. 2, 173-184, 1995.

[3] B. C. Damasceno and X. Xie "Petri nets and deadlock-free scheduling of multiple-resource operations," *IEEE Conf. on Systems, Man and Cybernetics,* 878-883, 1999.

[4] L. Ferrarini and M. Maroni, "A control algorithm for deadlock-free schedulings of manufacturing systems," *IEEE Conf. on Systems, Man and Cybernetics,* 3762-3767, 1997.

[5] Chu. Feng and X. Xiao-Lan "Deadlock analysis of petri nets using siphons and mathmatical programming," *IEEE tr. Robotics and Automation,* vol. 13. no. 6. 1997.

[6] S. Melzer and S. Romer "Deadlock checking using net unfoldings," *In Proc. of the Conf. on Computer-Aided Verfication,* CAV'97, 1997.

[7] T. Murata (1989). "Petri nets: properties, analysis an applications," *Proceedings of the IEEE,* 77(4), IEEE, USA, 541-580, 1989.

[8] H. H. Xiong and M. C. Zhou "Deadlock free scheduling of an automated manufacturing system based on petri nets," *In IEEE ICRA'97,* 945-950, 1997.

[9] H. Yoon and D. Lee, "Deadlock-free scheduling for automated manufacturing cells," *Proceeding of International Conference on Control, Automation Robotics and Vision,* 2000.

[10] M. A. Lawley, "Deadlock avoidance for production system with flexible routing," *IEEE tr. Robotics and Automation,* vol. 15, no. 3, pp. 497-509, 1999.

[11] M. P. Fanti, B. Maione, S. Mascolo, and B. Turchiano, "Event-based feedback control for deadlock avoidance in flexible production systems," *IEEE tr. Robotics and Automation,* vol. 13, no. 3, pp. 347-363, 1997.

[12] P. Jonghun and A. Spyros R. "Deadlock avoidance in sequential resource allocation systems with multiple resource acquisitions and flexible routings," *IEEE tr. Automatic Control,* vol. 46, no. 10, pp. 1572-1583, 2001.

송 유 진

1969년 6월 10일생. 1992년 창원대학교 컴퓨터공학과 졸업(학사). 1995년 창원대학교 대학원 컴퓨터공학과 졸업(석사). 2003년 창원대학교 대학원 컴퓨터공학과(공학박사). 1995년~2002년 창원대학교 강사. 2003년~현재 창원대학교 초빙교수. 관심분야는 패트리넷, 성능분석, 정보보호, 스케쥴링 연구.

이 종 근

1952년 4월 28일생. 1974년 숭실대학교 전자계산학과 및 동 대학원 공학석사. 1978년 고려대학교 경영대학원 경영학석사. 1987년~1990년 LSI/Univ. de Montpellier II 연구원 역임. 2002년 LCGI/Ecole Centrale Paris 전산학, 공학박사. 1983~현재 창원대학교 컴퓨터공학과 교수. 컴퓨터공학과장, 전산소장, 자연대부학장역임. 관심분야는 패트리 넷, 스케쥴링분석, 성능분석, 정보보호 관련 연구.