'93 한국자동제어학술회의논문집 (1993. 10. 20~22)

이진 페트리 네트의 메모리-베이스 구현을 기본으로 하는 프로그래머블 제어기에 관한 연구

° 장래혁, 박재현, 노갑선, 권욱현
서울대학교 제어계측 신기술연구소

A Programmable Controller Based on the Memory-based Implementation of a Binary-petri Net

° Naehyuck Chang, Jaehyun Park, Gab Seon Rho and Wook Hyun Kwon
Engineering Research Center for Advanced Control and Instrumentation, Seoul National University

ABSTRACT

For a fast evolution, a memory-based implementation of a petri net is discussed. A sub-class petri net model called B-petri net is suggested to make a memory-based implementation feasible in a large size application. The suggested B-petri net is a binary-petri net since only a binary-typed decision, fork and join are allowed. The application of a B-petri net is focused to a SFC(sequential function chart) program. The memory requirement, speed and computional load are compared with a petri net when they are implemented by a memory-based method.

1 서론

자동화 시스템에서 시퀀스제어가 차지하는 비중은 상당히 크며, 그 중에서도 프로그래머블 제어기(이하 PC)는 중요한 역할을 담당한다. 보다 복잡한 플랜트를 제어하기 위해서는 고성능의 PC가 필요하게 되는데, 고성능의 PC는 주어진 프로그램을 고속으로 처리하는 것은 물론, 정확한 동작을 하여야 하며, 사용자로 하여금 복잡한 프로그램을 보다 손쉽고 정확히 작성할 수 있도록 환경을 조성할 수 있어야 한다. 이러한 조건으로 볼 때 현재 가장 많이 사용되는 RLL(relay ladder logic) 프로그램은 고성능의 PC의 프로그램으로 그리 적합하지 않다고 보여진다[12, 5]. 플랜트를 보다 체계적이며 정확하게 제어 하려는 연구로 페트리네트(petri net), 스테이트 머신(state machine), 데이타플로우 그래프(dataflow graph) 등을 사용하여 시스템을 모델링하고 하드웨어를 제작하는 연구도 진행 되었다[12, 1, 6, 7, 8, 10].

이러한 시퀀스제어 시스템을 구현하는 방법으로는 해당 어플리케이션에 딱 들어맞는 전용하드웨어(customized hardwired logic)을 사용하는 것과 범용 마이크로 프로세서를 사용하는 방법 등도 생각할 수 있으나 본 연구에서는 비교적 빠른 속도와 규칙적인 구조, 유연성 등의 장점을 가진 메모리-베이스 구조에 대해서 연구한다. 메모리-베이스 구조는 커다란 LUT(look up table)를 만들어 LUT의 참조를 통하여 프로그램을 해석하는 방법이다[7, 1]. 이 방법에서는 프로그램이 LUT에 저장 되므로 제어 대상이 바뀔 때 이 테이블의 내용만을 바꿈으로써 변경이 가능하다. 이와같은 유연성을 가지면서도 전용의 하드웨어를 사용하는 시스템에 가까운 성능을 보여준다.

그런데 메모리-베이스 구조의 가장 큰 문제점은 메모리의 낭비에 있다. 메모리 테이블의 사이즈는 수용할 수 있는 모델의 크기에 따라서 기하급수적으로 증가하므로 큰 모델을 수용하는 LUT를 만들기는 쉽지 않다. 더구나 LUT의 대부분은 실제로는 아무 내용도 기록되지 않는다. 이러한 메모리의 낭비는 모델을 이진(binary) 타입으로 전개함으로써 낭비되는 부분을 크게 줄이고 따라서 큰 모델을 수용할 수 있도록 할 수 있다[7].

이진 전개시에도 문제점이 있는데, 원래 모델을 이진로 바꾸는 데서 발생하는 모델의 성질 변화와 수행 속도의 저하 등을 들 수 있다.

메모리-베이스 구현 방법은 스테이트 머신에서 먼저 연구 되었으며, 그 구조 또한 아주 간단 명료하게 된다. 스테이트 머신에서는 각 스테이트에서 다음으로 트랜지션할 스테이트를 결정하는 판단(decision)을 수행 하여야 하는데, 이를 이진 판단(binary decision)으로 바꾸어 메모리-베이스 구현을 하는 연구가 발표되었다. 이 연구에서는 스테이트 머신의 경우 이진 판단 구조를 사용할 때 생기는 영향을 분석 하였다[10]. 본 연구와 관련된 것으로 데이타플로우 그래프를 이용한 벙렬처리 PC에 관한 연구가 박재현 등에 의해서 진행 되었다. 이 연구는 데이타 플로우그래프의 메모리 베이스 구현에 관한 것인데, 프로그램을 이진 타입으로 하도록 되어있다. 따라서 큰 응용 프로그램일 경우에도 구현이 가능하며, 프로그램은 본래부터 이진 타입으로 하도록 되어있다[12].

페트리 네트의 메모리-베이스 구현으로는 이를 PC에 응용하는 것이 발표 되었다. 이 구현은 페트리네트를 그대로 구현하는 방법으로 모델의 플레이스(place) 수와 트랜지션(transition) 수의 곱에 해당하는 메모리 테이블이 필요하다. 따라서 아주 큰 사이즈의 메모리 테이블이 필요하며, 특히 테이블의 폭(width)이 매우 넓으므로 구현이 아주 어렵게 된다[8]. 이 시스템의 메모리의 사용량을 줄이는 연구에 관한 논문도 같이 발표되었는데 원래의 페트리네트를 서브페트리네트(sub-petri net)으로 분해하여 메모리의 사용량을 90%까지 줄이는 결과를 보여준다[9].

페트리네트를 메모리-베이스 구조로 구현하는데 역시 이진 전개를 사용하는 것이 커다란 응용 프로그램을 수용가능하도록 하는 방법이라 판단되어 본 연구에서는 이진-페트리네트를 재안하고 이를 실제로 구현할 때의 영향을 분석 하였다. 페트리네트를 이진 전개하려면 이진 포크(binary fork), 이진 조인(join), 이진 판단(binary)이 필요하다. 메모리 베이스 구현도 스테이트 머신보다 복잡하며, 한번의 트랜지션을 위해서는 여러번의 LUT 참조가 있어야한다. 본 연구와 관련된 선행 연구로 마크드 그래프의 이진 모델을 제안하고 이의 메모리-베이스 구현과 이에 따른 영향을 분석한 연구가 수행 되었다. 이진 마크드 그래프는 이진 포크와 이진 조인으로 재한되며, 마크드 그래프의 성질 변화 없이 변환 가능하다[1].

이진 판단 모델은 판단 조건을 순서대로 하나씩 검사하며, 이에 따라서 판단 조건 간에 우선 순위가 생기게 된다. 페트리네트의 판단 모델을 이진 판단으로 바꾸었을 때, 이와같은 우선 순위의 영향이 발생하여 모델의 성질을 다르게 하여서는 곤란하다. 본 연구에서는 이진 페트리네트를 PC의 프로그래밍 언어인 SFC(sequential function chart)에의 응용으로 재한하여 연구한다. SFC는 RLL 프로그램의 단점을 보완하려는 목적에서 재안 되었으며, 그 수학적인 근거는 페트리네트를 기본으로

한다[2]. SFC는 판단 모델이 우선순위를 가지도록 재정 되어 있으며, 또는 판단 조건이 서로 배타적(exclusive)하도록 되므로 이진 전개시, 이진 판단 모델의 적용이 무리가 없게 된다. 본 연구는 이진 마크드 그래프의 제안과 이의 메모리 베이스 구현에 관한 연구의 확장이며, 이 논문의 결과를 많이 이용한다.

2 장에서는 이진 패트리네트의 수학적인 모델을 제시하며, 3 장에서는 패트리네트의 메모리 베이스 구현에 대해서 설명한다. 4장에서는 제시된 모델의 분석이 이루어지고, 5 장에서는 메모리 베이스 구현과 SFC를 사용하는 PC의 적용이 논의된다. 마지막으로 6 장에서 본 논문을 결론 짓는다.

2 B-페트리네트(B-petri net)의 수학적 모델

B-페트리네트는 일반적인 페트리네트(ordinary petri net)의 서브 세트로서 이진 타입의 조인, 포크, 판단 모델만이 허용된다. 즉 모든 플레이스는 두개 이하의 출력 트랜지션을 가지며 모든 트랜지션은 두개 이하의 입력 플레이스와 두개 이하의 출력 플레이스를 가지는 것만이 허용된다.

지금부터 주어진 플랜트를 일반적인 페트리네트로 모델한 것을 $C = (P, T, I, O)$, $|T| = m$, $|P| = n$이라 하자. C는 일반적인 페트리네트이므로 플레이스 중에서는 출력 트랜지션의 수가 세개 이상인 것이 존재 할 수 있으며, 트랜지션 역시 세개 이상의 입, 출력 플레이스를 가지는 것이 존재 할 수 있다. C와 같은 기능을 하는 B-페트리네트의 모델을 $\mathcal{B} = (P', T', I', O')$라 하면, $\mathcal{B}$는 변환 작업을 통해서 C와 같은 기능을 가질 수 있게 된다.

B-페트리네트는 다음과 같이 정의되며 페트리네트의 정의를 이용한다[3].

정 의 1 *B-페트리네트 $\mathcal{B} = (P', T', I', O')$는 다음과 같은 제약 조건을 가지는 페트리네트이다. 즉,*

$$\begin{aligned} |I'(t_j)| &\leq 2, \\ |O'(t_j)| &\leq 2, \\ |O'(p_i)| &\leq 2. \end{aligned}$$

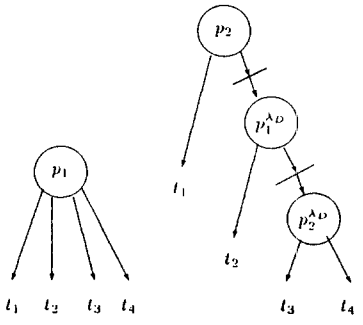
여기서,

P'	=	$\{p_1, p_2, ..., p_n\}$	은 유한 크기의 플레이스의 집합,
T'	=	$\{t_1, t_2, ..., t_m\}$	은 유한 크기의 트랜지션의 집합,
$I'(t)$	=	$\{p \mid (p,t) \in F\}$	은 트랜지션 t의 입력 플레이스,
$O'(t)$	=	$\{p \mid (t,p) \in F\}$	은 트랜지션 t의 출력 플레이스,
μ'	=	$(\mu(p_1), ..., \mu(p_n))^T$	은 $n \times 1$ 벡터로 플레이스의 마킹 *(marking)* 이다.

단, $m' = |T'|$, $n' = |P'|$. ■

페트리네트 C를 B-페트리네트로 변환하는 과정을 크게 두가지로 이진 포크, 이진 조인 변환과 이진 판단 변환으로 나누어 생각한다. 이진 포크 변환과 이진 조인 변환은 세개 이상의 입력 플레이스와 세개 이상의 출력 플레이스를 가진 트랜지션을 두개 이하의 입력 플레이스와 두개 이하의 출력 플레이스를 가지는 이진 타입의 조인과 포크의 다단 결합으로 바꾸는 작업이다[1]. 이 때 물론 모델의 성질에는 변화가 없어야 한다. 이 변환은 심플 리덕션 규칙(simple reduction rule)에 포함되는 것이므로 성질의 변화는 없게된다[4]. 이진 포크와 조인 변환은 마크드 그래프의 경우와 같으며, 마크드 그래프를 B-마크드 그래프로 변환하는 것과 같다[1].

판단 모델의 경우는 이진 변환을 하는 경우 판단의 우선순위가 생기게 된다. 그림 2의 (a)는 페트리네트의 판단 모델이며, 마킹은 t_1, t_2, t_3, t_4 중 한 곳으로 옮겨가는데 모두 같은 우선순위를 가진다. (b)의 경우는 B-페트리네트의 경우인데, 이 경우 같은 우선순위라 할 수 없으며 따라서 (a)와 (b)는 같은 모델이라 할 수 없다.



(a) 페트리네트 C (b) B-페트리네트 $\mathcal{B}$

그림 2 패트리네트와 B-페트리네트의 판단 모델과 우선순위

제안된 B-페트리네트는 원래의 페트리네트 모델이 판단시 우선 순위를 가지고 있어서 이를 변환하더라도 같은 우선순위를 가진다는 가정하에서 사용된다. 즉 변환전의 페트리네트 C의 판단 모델이 원래부터 왼편에서 오른편으로 우선 순위를 가지고 판단 된다고 하면 이진 판단 변환을 거치더라도 같은 모델이 된다는 가정이다. 본 연구에서 B-페트리네트를 적용하는 SFC 프로그램에서는 이와같은 우선순위가 적용되는 모델이므로 B-페트리네트를 적용하더라도 무리가 없게 된다[1].

이진 포크 변환과 이진 조인 변환, 이진 판단 변환을 통해 원래의 패트리네트를 B-페트리네트로 변환하게 되면 원래의 모델보다 더 많은 수의 플레이스와 트랜지션을 가지게 된다. 이 때 늘어나는 플레이스와 트랜지션을 널-플레이스(null place) P^{λ}와 널-트랜지션(null transition) T^{λ}라 하며, 따라서 플레이스와 트랜지션의 집합인 P와 P', T와 T'은 $P \subseteq P'$, $T \subseteq T'$의 관계를 가진다. 즉 $P \bigcup P^{\lambda} = P'$, $T \bigcup T^{\lambda} = T'$ 이 된다. 또 플레이스의 마킹을 나타내는 벡터 μ와 μ'도 $\mu' = (\mu^T \ \mu^{\lambda T})^T$ 가 된다. 널-플레이스와 널-트랜지션은 이진 조인, 이진 포크, 이진 판단 모델에서 각각 존재하는 것의 합집합이 되며 이들은 다음과 같이 표시한다[1]

$P_t^{\lambda f}$	트랜지션 t의 이진 포크 변환시 생기는 널-플레이스 집합.
$P_t^{\lambda j}$	트랜지션 t의 이진 조인 변환시 생기는 널-플레이스 집합.
$T_t^{\lambda f}$	트랜지션 t의 이진 포크 변환시 생기는 널-트랜지션 집합.
$T_t^{\lambda j}$	트랜지션 t의 이진 조인 변환시 생기는 널-트랜지션 집합.
$P_t^{\lambda D}$	플레이스 p의 이진 판단 변환시 생기는 널-플레이스 집합.
$T_t^{\lambda D}$	플레이스 p의 이진 판단 변환시 생기는 널-트랜지션 집합.

페트리네트 C에서 어떤 트랜지션이 파이어링(firing) 되는 것과 같은 동작이 B-페트리네트에서 이루어지려면 이를 이진 변환한 모델의 트랜지션들이 모두 파이어링이 일어나야 한다. 이 때 페트리네트 C의 임의의 트랜지션에 해당하는 B-페트리네트 $\mathcal{B}$의 트랜지션들의 집합을 등가 트랜지션 집합(equivalent transition set) 이라고 정의 한다[1]. 이 등가 트랜지션 집합은 서로의 교집합이 공집합이다(disjoint). B-페트리네트의 상대적인 계산량 및 속도 등은 이와같은 등가 트랜지션 집합과 이에 해당하는 마킹의 검사 및 갱신등을 기준으로 행해지게 된다.

3 페트리네트의 메모리-베이스 구현

패트리네트의 메모리-베이스 구현의 용량은 몇개의 플레이스와 몇개의 트랜지션으로 모델된 페트리네트까지 수용이 가능한 가로 말할 수 있다. 여기서는 m 개의 트랜지션과 n개의 플레이스를 가지는 페트리네트 C를 구현하는 것으로 한다. 페트리네트의 메모리-베이스 구현은 마크드 그래프의 메모리 베이스 구현에서 플레이스의 출력 트랜지션 테이블에 멀티플렉서(multiplexer)와 이를 제어하는 로직이 추가된 형태이며 그림 3의 구조를 가진다[1].

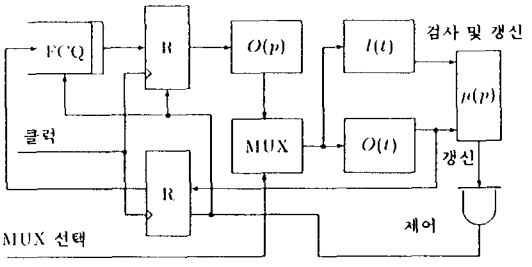


그림 3 페트리네트의 메모리-베이스 구현 구조

FCQ는 초기마킹 (initial marking)을 가진 플레이스가 저장되는 큐(queue)이다. 큐형태를 가지므로 페트리네트가 수행(evolution) 되는 순서가 FCQ에 저장되는 순서로서 임으로 결정되지만, 우리가 페트리네트의 논리적인(logical) 동작에 관심이 있고 시간적인 고려를 하지 않으면 같은 결과를 얻는다고 말할 수 있다. 따라서 구현하기 쉬운 큐의 형태를 취하여도 가능하다. 초기마킹이 저장된 FCQ에는 파이어링이 일어나는 대로 계속해서 파이어링이 일어난 트랜지션의 출력 플레이스, 즉 새롭게 마킹이 전달된 플레이스가 계속 들어온다. 만일 더 이상의 파이어링이 일어나지 않고 FCQ가 비게되면(empty) 이 페트리네트는 라이브(live) 하지 않은 상태가 된 것이된다. R은 레지스터를 말하며 메모리와 같은 조합로직(combinational logic)의 피드백을 막는다. $O(p)$는 각 플레이스들의 출력 트랜지션들을 저장하고 있으며 이 테이블의 출력인 출력 트랜지션들은 멀티플렉서(MUX)와 이를 선택하는 신호로부터 하나의 트랜지션으로 바뀌게 된다. 이 동작이 플레이스에서의 판단 동작이 된다. $I(t)$와 $O(t)$는 플레이스들의 입, 출력 플레이스 테이블이다. 이들은 멀티플렉서로부터 선택된 트랜지션이 파이어링 가능한가(enable)를 검사하기 위해 이 트랜지션의 모든 입력 플레이스들을 찾는 일과, 파이어링이 일어난 후에 이 트랜지션의 모든 입, 출력 플레이스들의 마킹을 갱신하기 위해 모든 입, 출력 플레이스를 찾는데 사용된다. 이들의 출력은 마킹의 정보가 저장되어있는 $\mu(p)$ 테이블의 입력으로 사용된다.

이들 테이블의 크기 및 차원(dimension)은 다음과 같다.

$\mu(p)$은 $n \times 1$ 크기이며 입력은 플레이스 출력은 마킹.

$O(p)$은 $n \times m$ 크기이며 입력은 플레이스 출력은 트랜지션들.

$I(t)$은 $m \times n$ 크기이며 입력은 트랜지션 출력은 플레이스들.

$O(t)$은 $m \times n$ 크기이며 입력은 트랜지션 출력은 플레이스들.

FCQ 너비 1의 큐이며 입, 출력은 플레이스.

B-페트리네트의 경우는 메모리 테이블의 차원과 사이즈가 다르게 된다. m'개의 트랜지션과 n'개의 플레이스를 가지는 B-페트리네트를 고려하면 다음과 같다. 이 때 페트리네트 C와 B-페트리네트 B의 경우를 비교하기 위해서는 m과 m', n과 n'의 관계를 파악하면 되는데 이는 4장에서 언급한다.

$\mu(p)$은 $n' \times 1$ 크기이며 입력은 플레이스 출력은 마킹.

$O(p)$은 $n' \times 2$ 크기이며 입력은 플레이스 출력은 트랜지션들.

$I(t)$은 $m' \times 2$ 크기이며 입력은 트랜지션 출력은 플레이스들.

$O(t)$은 $m' \times 2$ 크기이며 입력은 트랜지션 출력은 플레이스들.

FCQ 너비 1의 큐이며 입, 출력은 플레이스.

그림 3의 구조로 페트리네트를 수행시키는 알고리듬은 다음과 같다[1].

step 1 초기상태로 초기마킹이 있는 플레이스들을 FCQ에 넣는다.

step 2 FCQ에서 플레이스 p를 꺼낸다.

step 3 FCQ에서 꺼낸 플레이스 p를 가지고 $O(p)$에서 모든 출력 트랜지션을 꺼낸다.

step 4 판단 조건에 의해서 MUX로부터 여러개의 출력 트랜지션중 하나의 t를 선택한다.

step 5 t의 입력 플레이스를 $I(t)$에서 모두 찾는다.

step 6 만일 입력 플레이스의 마킹이 모두 1 이면 즉, $\forall p \in I(t)$, $\mu(p) = 1$ 이면 step 7로 아니면 step 2로 간다.

step 7 t의 모든 출력 플레이스를 $O(t)$에서 찾는다.

step 8 t의 모든 입, 출력 플레이스의 마킹을 갱신한다.

step 9 t의 모든 출력 플레이스를 FCQ에 넣는다.

step 10 step 2로 간다.

마크드 그래프의 경우와 마찬가지로 페트리네트에서도 대부분의 플레이스들이 특정 트랜지션의 입력 플레이스나 출력 플레이스가 아닌 경우가 많고, 대부분의 트랜지션들이 특정 플레이스의 출력 트랜지션이 아닌 경우가 보통이다. 따라서 페트리네트의 메모리-베이스 구현의 경우에는 메모리 테이블의 대부분이 비어있게 된다. 또, 이러한 테이블은 폭이 매우 넓으므로 실제로 구현하기도 매우 까다로우며, 마킹 테이블 같은 경우에도 이를 동시에 갱신 하기 위해서는 테이블의 길이가 아닌 폭을 넓게 해야 하므로 역시 현실적으로 구현이 용이하지 않다.

4 B-페트리네트의 분석

B-페트리네트를 페트리네트 대신에 사용하여 메모리-베이스 구현을 하는 경우, 메모리의 사용량이나 동작 속도, 계산량등이 성능 비교를 위한 요소가 된다. 같은 동작을 하는 페트리네트 C, B-페트리네트 B의 성능 비교를 이 세가지 요소의 분석을 통해서 언급한다. 이 세가지 요소는 페트리네트 C가 B-페트리네트 B로 변환 되었을 때 생기는 널-플레이스와 널-트랜지션의 수에 따라 결정된다. 널-플레이스와 널-트랜지션의 수는 모델의 형태에 따라서 변화가 심하며, 여기서는 상한과 하한을 계산한다.

4.1 메모리 사용량

메모리 사용량은 페트리네트를 메모리 베이스 구현할 때 가장 큰 비중을 차지하는 메모리 테이블에 의해서 결정된다고 볼 수있다. $I(t)$, $O(t)$, $O(p)$가 이에 해당하며 이의 크기는 $m \times n$이므로 다른 테이블에 비하여 모델이 커짐에 따라서 급격히 크기가 증가한다. 이 테이블 중 $I(t)$와 $O(t)$는 이진 조인과 포크 변환에 의해서 $m' \times 2$로 바뀌며, $O(p)$는 이진 판단 모델에 의해서 $n' \times 2$로 바뀐다. 이 때, m'과 n'의 상한과 하한을 계산하기 위해서 다음의 선행정리 1을 사용한다[1].

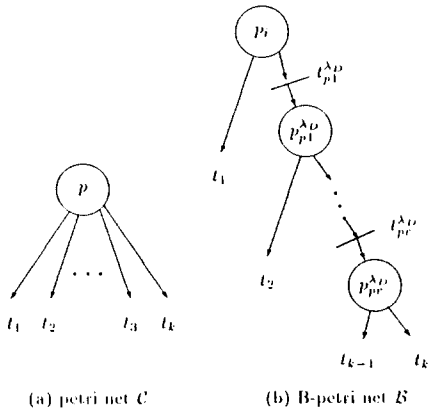
선행정리 1 입력 플레이스의 수가 k(단, $k \geq 2$)개인 조인 모델을 이진 조인 모델로 바꾸면 널-플레이스의 수와 널-트랜지션이 $k-2$개 각각 생긴다. ■

이진 판단 모델의 경우는 그림 4.1에 나타나 있으며, 이 경우의 널-플레이스와 널-트랜지션의 발생은 선행정리 2와 같다.

선행정리 2 출력 플레이스가 k(단, $k \geq 2$)개인 플레이스 p의 판단 모델을 이진 판단 모델로 바꾸면 $k-2$개의 널-플레이스와 널-트랜지션이 각각 생긴다.

증명: 귀납법으로 증명한다.

만일 $k = 2$ 이면, $|P_t^{AD}| = 0$.
만일 $k = 3$ 이면, $|P_t^{AD}| = 1$.



(a) petri net C (b) B-petri net B

그림 4.1 이진 판단 변환시 널-플레이스와 널-트랜지션의 발생

만일 $k = 4$ 이면, $|P_t^{\lambda p}| = 2$.

만일 $k = i$ 에서 $|P_t^{\lambda p}| = i - 2$라 가정하자. 이 때 출력 플레이스가 하나 더 늘어나면 하나의 널-플레이스와 하나의 널-트랜지션이 더 필요하게 된다. 즉, $k = i + 1$일 때 $|P_t^{\lambda p}| = i - 1$.

■

페트리네트 C를 B-페트리네트 B로 바꾸는 경우 발생하는 널-플레이스와 널-트랜지션의 상한과 하한을 선행정리 2과 1를 사용하여 계산한다. 하한의 경우는 애초부터 C가 B-페트리네트의 제약 조건에 맞도록 모든 플레이스의 출력 트랜지션의 수와 모든 트랜지션의 입, 출력 플레이스의 수가 두개 이하인 경우이다. 이 경우는 C와 B는 완전히 같으므로 널-플레이스와 널-플레이스가 하나도 생기지 않는다. 반대로 상한의 경우는 C의 모든 플레이스가 모든 트랜지션을 출력 트랜지션으로 가지고, 모든 트랜지션의 입, 출력 플레이스가 모든 플레이스인, 플레이스와 트랜지션간의 링크(link)가 완전한 연결을 가지는(completly connected) 경우이다. 물론 이런 경우는 없겠지만 이 경우 C를 B로 바꿀 경우 가장 많은 널-플레이스와 널-트랜지션이 생긴다.

정리 1 페트리네트 C와 이의 이진 변환인 B-페트리네트 B에서, C를 B로 변환시 생기는 P^λ와 T^λ의 크기는 다음과 같은 상한을 가진다. 단, $m = |T|$, $n = |P|$

$$\begin{cases} 0 \le & |P^\lambda| & \le 3mn - n - 4m \\ 0 \le & |T^\lambda| & \le 3mn - n - 4m \end{cases}$$

증명: 하한의 경우는 자명하므로 상한을 생각한다. 각 n개의 플레이스는 m개의 출력 플레이스를 각각 가지고 m개의 트랜지션은 모두 n개의 입력 플레이스와 n개의 출력 플레이스를 가진다. 이들에 의해서 생기는 널-플레이스와 널-트랜지션은 모두 교집합이 공집합이므로 *(disjoint)* 각각 갯수의 합과 같다. 즉,

$$|P^{\lambda p}| = \sum_{i=1}^{n} |P_{t_i}^{\lambda p}| = (m-2)n$$

$$|P^{\lambda t}| = \sum_{i=1}^{n} |P_{p_i}^{\lambda t}| = (n-2)m$$

$$|P^{\lambda *}| = \sum_{i=1}^{n} |P_{p_i}^{\lambda *}| = (n-2)m$$

따라서,

$$|P^\lambda| = (m-2)n + 2(n-2)m = 3mn - 2n - 4m$$

마찬가지로 하면,

$$|T^\lambda| = (m-2)n + 2(n-2)m = 3mn - 2n - 4m$$

■

결국 B-페트리네트 B의 플레이스의 총 수와 트랜지션의 총 수는 정리 1의 널-플레이스와 페트리네트 C의 플레이스 수를 더한 것이 된다. 즉,

$$n \le n' = |P'| = |P^\lambda| + |P| \le 3mn - n - 4m.$$

또, 트랜지션의 수는,

$$m \le m' = |T'| = |T^\lambda| + |T| \le 3mn - 2n - 3m.$$

이 된다.

4.2 계산량 및 수행 속도

계산량은 페트리네트의 하나의 트랜지션을 파이어링 하기위해서 입력 플레이스를 찾고 이의 마킹을 검사한 후에 파이어링이 일어난 후에 입력 플레이스의 마킹과 출력 플레이스의 마킹을 갱신하는 데 걸리는 메모리 참조 횟수와 비교 횟수로 정의한다. 페트리네트 C의 경우 입력 플레이스를 찾고, 마킹을 검사하고, 입, 출력 플레이스의 마킹을 갱신하는 데 n개의 값을 동시에 4 번 참조 하여야 한다. 마킹을 비교하는데는 n번수의 연산이 1 번 필요하다. B-페트리네트 B의 경우에는 항상 2 개의 값만을 동시에 메모리에서 참조하며, 대신에 횟수는 페트리네트에 비해서 많게 된다. 마킹의 비교시에도 2 변수의 비교로 이루어지며 역시 횟수는 많아진다. 이들 횟수는 모델에 따라서 변하므로 역시 상한과 하한으로 구분할 수 있다.

수행 속도는 페트리네트 C의 한개의 트랜지션이 파이어링 되는 속도와 이와 등가인 B-페트리네트의 트랜지션들이 모두 파이어링 되는 속도로 비교가 된다. 트랜지션의 횟수는 역시 하한은 1 번이나, 물론 모델에 따라서 변한다.

그런데 계산량과 수행속도는 비교하는 횟수나 파이어링 하는 횟수는 B-페트리네트 경우가 더 많지만 한번 비교하는데 필요한 비용과 한번 트랜지션을 수행하는데 필요한 비용은 B-페트리네트가 월등히 적으므로, 비현실적으로 복잡한 모델이라도 같은 비용에서 수행할 수 있는 속도는 비슷하다고 볼 수있다. 뿐만 아니라, 실제로 사용되는 모델은 하한에 매우 가까우므로 수행 속도는 대부분의 경우 더 빠르다고 볼 수 있다.

5 SFC 프로그램과 PC

본 장에서는 B-페트리네트의 응용으로 RLL 프로그램을 대치할 수 있게 새로이 제정된 PC 프로그래밍 언어인 SFC에 대해서 언급한다. SFC는 수학적인 모델로는 페트리네트를 근거로 하고 있으며, 보다 효율적인 제어기 설계를 할 수 있도록 되어있다. SFC를 수행 시키는데는 RLL 프로그램 처럼 일반 마이크로 프로세서 등을 써서도 가능 하지만 페트리네트의 성질을 살린 메모리-베이스 구현을 이용하면, 빠르고 효율적인 구현이 가능하다. 실제 응용에 사용되는 큰 크기의 프로그램의 경우도, B-페트리네트를 사용하면 충분히 구현이 가능하게된다.

SFC의 수학적인 모델은 다음과 같이 정의된다[2].

정의 2 *SFC*의 수학적인 모델은 4 개의 튜플*(tuple)*을 가진 디렉티드 그래프이다 *(directed graph)*. 즉, $< X, T, L, X_0 >$에서,

X	=	$(x_1, \ldots, x_n)$	유한의 스텝의 집합,
T	=	$(t_1, \ldots, t_m)$	유한의 트랜지션의 집합,
L	=	$(l_1, \ldots, l_h)$	스텝과 트랜지션을 연결하는 디렉티드 링크,
X_0	$\subset$	X	초기스텝.

또, 스텝은 명령*(command)*과 동작*(action)*과 관련되어 있고, 트랜지션

은 로직 트랜지션 조건(*logic transition condition*)과 관련 되어있다. ■

SFC에서의 스텝은 페트리네트에서 플레이스에 해당하고 트랜지션은 트랜지션에 해당한다. SFC의 판단 모델과 동시에 수행되는 병렬(parallel) 분기와 이들이 다시 합치는 모델은 페트리네트와 같으므로 페트리네트에 바로 적용이 가능하다. SCF가 스텝과 트랜지션에 명령등을 함축하고 있으므로 이에 적용할 때는, 튜플(tuple)을 늘려서 트랜지션 조건과 각 스텝에서의 동작을 명시해야 한다.

SFC를 적용하기 위해서 B-페트리네의 튜플을 늘려서 이를 정의하면 다음과 같다.

정의 3 *B*-페트리네트에 *SFC*를 적용하기 위해서 튜플을 늘리고 이를 B^P-페트리네트라 한다. 즉,

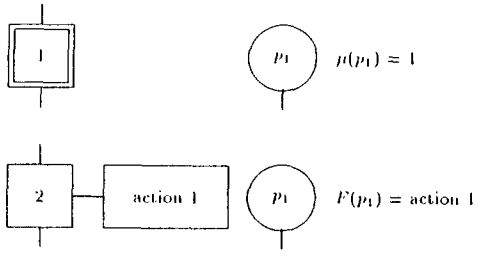
$$P = (P', T', I', O', \mu', F, D),$$

그런데,

$F(t)$ = $\{f|(t, f)\}$ 트랜지션에서 정의된 트랜지션 조건 함수,

$D(p)$ = $\{d|(p, d)\}$ 플레이스에서 정의된 명령과 오퍼랜드. ■

SFC 프로그램은 거의 1:1 로 B-페트리네트로 변환 가능하다. 그림 5에는 초기 스텝의 처리와 스텝에서 명시되는 동작의 구현 방법을 볼 수 있다. 초기 스텝은 초기 마킹을 메모리-베이스 구현의 FCQ에 넣음으로써 가능하다. 각 스텝에 할당된 명령은 플레이스의 함수인 $D(p)$에 의해서 명시된다. SFC의 경우 한 스텝에 할당된 명령이 단순한 한의 명령이 아니고 하나의 서브루틴인 경우도 가능하다. 이런 경우 메모리-베이스 구현의 B^P-페트리네트는 이러한 일련의 명령(서브루틴)을 코디네이트(coordinate)하는 제어기로 사용된다.



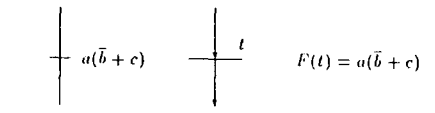
(a) SFC 프로그램 (b) B^P-페트리네트

그림 5 SFC 프로그램과 B^P-페트리네트 (1)

페트리네트의 파이어링 규칙은 인에이블 되어있는 트랜지션은 파이어 가능(may fire) 하다는 애매한 정의가 되어있는데 실제로 제어기를 구현할 때는 이와같은 애매한 정의는 사용할 수 없다. SFC에서는 파이어링을 인에이블과 클리어(clear)로 정의를 하고 마킹 혹은 토큰이 트랜지션의 입력 스텝에 모두 존재하고 별도로 지정된 트랜지션 조건이 참인 경우에는 마킹을 넘겨주는 클리어 동작을 하도록 되어있다[2]. 즉, 인에이블 되고 트랜지션 조건을 만족하면 이를 바로 클리어 시키면 된다.

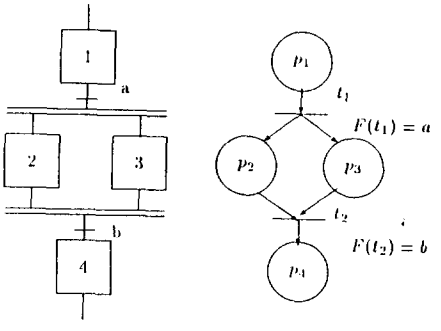
SFC의 트랜지션 조건은 방정식이나 기호나 텍스트 등으로 이를 명시해 주면 된다[2]. B^P-페트리네트에서도 마찬가지로 $F(t)$의 트랜지션의 함수로 명시한다.

SFC의 트랜지션의 동시 클리어링은 두줄의 라인으로 표시하며, 페트리네트의 포크와 같다. 또, 동시에 클리어링이 된 후에 다시 두줄의 라인으로 표시된 것으로 모이면, 이는 페트리네트의 조인과 같은 기능을 한다. 한줄의 라인은 페트리네트의 판단과 같으며, 스텝으로의 모이는 한줄은 페트리네트와 같이 논리합(OR)으로 동작을 한다[2]. 이들의 변환은 그림 5에서 볼 수 있다.



(a) SFC 프로그램 (b) B-페트리네트

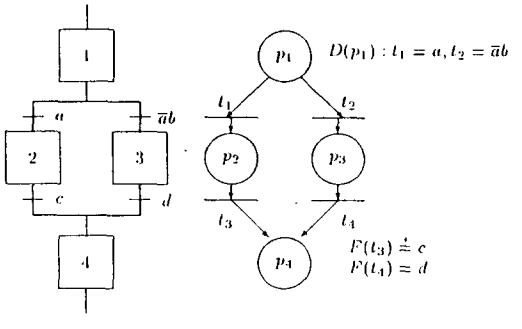
그림 5 SFC 프로그램과 B^P-페트리네트 (2)



(a) SFC 프로그램 (b) B-페트리네트

그림 5 SFC 프로그램과 B^P-페트리네트 (3)

동시에 수행되는 스텝이 아니고 여러개의 시퀀스 중에서 하나를 선택하는 것은 한줄의 라인으로 표시한다. 선택(selection)은 이 중 하나의 시퀀스가 되며, 하나만 선택이 되도록 선택의 조건을 우선순위가 있도록 정리하여 표시하면 된다[2]. 그림 5에 이 경우의 변환이 나와 있다.



(a) (b)

그림 5 SFC 프로그램과 B^P-페트리네트 (4)

SFC 프로그램을 수행하는데 있어서 보통의 마이크로 프로세서를 사용하지 않고 전용의 하드웨어를 사용하는 것은 수행 속도의 면에서 매우 유리하다. 메모리-베이스 하드웨어를 사용하면, SFC의 프로그램을 그대로 옮길 수 있으며, 마이크로 프로세서 때와 같이 모델이 바뀔 때 코우딩(coding)을 다시해야 하는 부담을 줄일 수 있다. SFC 프로그램이 PC의 프로그램 언어이므로 이를 수행하는 하드웨어는 프로그램의 내용이 바뀌더라도 손쉽게 대처할 수 있어야 하며, 이런 것을 고려할 때에 메모리-베이스 구현은 매우 적합하다고 볼 수있다. 제안된 B-페트리네트는 실제 사용될 수 있는 길이의 SFC 프로그램를 수행하는 메모리-베이스 하드웨어를 구현 가능하게 한다. 이진 구조의 B-페트리네트 메모리-베이스 하드웨어는 일반의 컴퓨터 하드웨어와 같이 고정된 비스폭을 가지며, 용량을 확장할 때에도 비스폭을 고정 시킨채로 메모리의 깊이만

을 늘이면 가능하다. 반면 페트리네트의 메모리-베이스 하드웨어는 버스 폭 자체가 매우 넓을 뿐아니라 용량을 늘릴 때에는 버스 폭 자체를 늘려야하므로 매우 비용이 많이 든다.

6 결론

본 연구에서는 페트리네트를 메모리-베이스 구조를 사용하여 구현할 때 생기는 문제점을 해결하기 위해서 B-페트리네트를 제안하고 이를 사용할 경우의 성능을 분석 하였다. 메모리-베이스 구조의 페트리네트 구현은 마크드 그래프의 경우를 확장 하였고, 이 때 사용되는 이진 판단 모델의 제약을 고려하여 B-페트리네트의 응용을 PC의 SFC 프로그램으로 제한 하였다.

SFC 프로그램은 RLL 프로그램에 비하여 매우 많은 장점을 가지고 있는데, 그 수학적인 모델을 페트리네트를 기반으로 하였으므로 보통의 마이크로 프로세서를 사용하여 이를 구현 하지않고 전용의 하드웨어를 사용하면 많은 이득이 있다. B-페트리네트는 SFC 프로그램에 매우 적합하며 대부분의 SFC 프로그램은 B-페트리네트 변환시 매우 좋은 성능을 보여준다. B-페트리네트를 사용한 메모리-베이스 구조는 구현하기 쉬우며 확장시 무리가 없는 장점으로 실제 시스템에 적용이 가능하다.

References

[1] Naehyuck Chang, Jachyun Park, Kyeonghoon Koo and Wook Hyun Kwon, "Memory-based Implementation of a Petri Net and its application to a Programmable Controller," *IECON '93*.

[2] *International Standard IEC 848, First Edition*, 1988.

[3] James L. Peterson, *PETRI NET THEORY AND THE MODELING OF SYSTEMS*, Prentice-Hall, Inc., 1981.

[4] Tadao Murata, "Petri Nets: properties, Analysis and Applications," *Proceedings of IEEE*, vol. 77, no. 4, Apr. 1989.

[5] Albert Falcione and Bruce H. Krogh, "Design Recovery for Relay Ladder Logic," *IEEE Control Systems*, Apr. 1993.

[6] A Di Stefano and O Mirabella, "A fast sequence control device based on enhanced Petri Nets," *Microprocessors and Microsystems*, vol. 15 no. 4, May 1991.

[7] Paul C. Baracos, Robert D. Hudsin, Luis J. Vroomen and Paul J. A. Zsombor-Murray, "Advances in Binary Decision Based Programmable Controllers," *IEEE Trans. Industrial Elec.*, vol. 25, no. 3, Aug. 1988.

[8] Hideki Murakoshi, Miki sugiyama, Guojun Ding, Tatsuya Onmi, Takashi Sekiguchi and Yasunori Dohi,"A High Speed Programmable Controller Based on Petri Net," *IECON '91*.

[9] Hideki Murakoshi, Miki Sugiyama, Guojun Ding, Tatusya Onmi, Takashi Sekiguchi and Yasunori Dohi,"Memory Reduction of Fire Unit for Petri Net Controlled Multiprocessor," *IECON '91*.

[10] Lee D. Coraor, Paul T. Mulina and Orlando A. Morean, "A General Model for Memory-Based Finite-State Machines," *IEEE Trans. Comput.*, vol. C-36, pp. 175-184, 1987.

[11] T.Murata and J. Y. Koh, "Reduction and expansion of live and safe marked graphs ," *IEEE Trans. Circuits Syst.*, vol. 27, no. 1, pp. 51-76, Aug. 1983.

[12] Jachyun Park, Naehyuck Chang, Gab Seon Rho and Wook Hyun Kwon "Implementation of Parallel Logic Solving Algorithm for PLC Based on Dataflow Architrcture," *Preprint of the '92 IFAC Workshop on Algorithm and Architectures for Real-time Cotrol*, Aug. 1992.