Slice 모델을 이용한 유한상태머신의 트랜지션 축약 알고리즘

(A Transition Reduction Algorithm of Finite State Machines using Slice Models)

이 우 진 †

(Woo Jin Lee)

요 약 실생활과 밀접한 환경에서 컴퓨터 시스템의 사용이 늘어남에 따라 이러한 시스템들의 신뢰성, 안전성 등을 철저히 확인할 수 있는 검증 기술의 필요성이 한층 증가되고 있다. 유한상태머신은 이러한 시스템 모델링 및 분석에 활용되는 다양한 정형적 기법의 기본 모델로 널리 이용되고 있으며 효율적인 분석을 위해 내부 이벤트에 대한 추상화 기법이 제공된다. 하지만 기존의 추상화 방법은 유한상태머신 모델을 생성한 후에 적용될 수 있으므로 상태폭발에 효과적으로 대처하지 못하는 문제점이 있다. 이 연구에서는 유한상태머신 정보를 효과적으로 표현할 수 있는 새로운 표기법으로, 병행적 특성을 나타낼 수 있는 slice 모델 개념을 제시하고 이를 근간으로 내부 트랜지션을 축약하는 추상화 알고리즘을 제시한다. 기존의 유한상태머신 기반의 추상화 방법은 모든 상태를 나열하여야만 분석할 수 있는데 반해, 제안된 추상화 방법은 필요한 분석 공간만을 부분적으로 생성함으로 시간 및 공간적인 관점에서 효율적이다.

키워드 : 유한상태머신, 상태폭발, Slice 모델, 트랜지션 축약 알고리즘

Abstract As the usage of computer systems is increasing in our lives, the reliability and safety of these systems need to be thoroughly checked through the verification techniques. As a basic formalism for several modeling methods, the finite state machine (FSM) is widely used in specification and verification of system models. And there is a technique for abstracting internal events of FSM in order to effectively analyze the system. However, this technique does not handle the state explosion problem since it can be applied after completely generating all the state space of the system. In this research, we provide a new approach for efficiently representing concurrent properties of FSM, the slice model and provide an efficient transition reduction method based on the slice model. Our approach is effective in time and space perspective since it is performed by partially generating the needed system states while the existing abstraction technique can be applied to all the system states.

Key words : Finite State Machine, State Explosion, Slice Model, Transition Reduction Algorithm

1. 서 론

현재 실생활에서 쉽게 접할 수 있는 정보가전, 휴대폰, 자동차 등에 내장형 컴퓨터 시스템이 널리 사용되고 있으며 앞으로 다가올 유비쿼터스 컴퓨팅 환경에서는 더 많은 분야에서 컴퓨터 시스템이 활용될 것으로 예상된다. 이러한 소프트웨어 시스템을 보다 안전하고 신뢰성 있게 활용하기 위해서는 시스템 개발 과정에서 모델링과 검증 과정을 통해 신뢰성, 안전성 등을 분석하여야 할 뿐만 아니라, 최종 시스템에서도 다양한 검사 및 확인 과정들을 수행하여야 한다.

소프트웨어 시스템의 모델링 및 분석을 수행하기 위해서는 시스템의 행위를 유한상태머신[1], Statecharts [2], Petri nets[3], Labeled Transition Systems[4] 등과 같은 정형적 기법으로 모델링하여 나타낸 다음, 교착상태와 안전성 등의 여러 시스템의 속성들을 검사하여

· 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다. (UD060048AD)

† 종신회원 : 경북대학교 전자전기컴퓨터학부 교수
woojin@knu.ac.kr

논문접수 : 2007년 9월 4일
심사완료 : 2007년 12월 26일

정보과학회논문지: 소프트웨어 및 응용 제35권 제1호(2008.1)

야 한다. 비록 다양한 정형적 기법들이 존재하여 시스템 특성에 따라 모델링할 수 있지만 시스템 분석 시에는 유한상태머신으로 변환하여 분석하는 경우가 많다. 예를 들어, Petri net 모델의 경우는 유한상태머신의 일종인 도달성 그래프를 생성하여 시스템 속성을 분석한다. 유한상태머신은 단순한 구조를 가지므로 분석하기 용이하지만 시스템의 규모가 커지면 시스템 상태의 수가 기하급수적으로 증가하는 상태폭발 문제를 가지는 단점이 있다.

이러한 상태 폭발 문제를 극복하기 위해 지금까지 다양한 연구가 진행되어 왔다. 시스템 상태의 대칭성[5] 또는 동치성[6]을 이용한 축약 방법, 서브 시스템으로 구성된 모델을 계층적으로 취합하면서 분석하는 합성적 방법[7], 병행적인 이벤트들에 대해 전체순서관계로 나열하지 않고 부분 순서관계로 시스템 상태를 나열하는 부분순서 축약 방법[8], 시스템 상태를 효율적으로 인코딩하여 저장하는 방법인 비트-상태 해싱과 해쉬 축약 방법[9], 안전한 방법으로 방문한 상태를 삭제하여 효율적인 동적 메모리 관리 기법을 제공하는 상태 공간 캐싱과 sweep-line 방법[9], 그리고 유한상태머신의 관심이 없는 트랜지션을 축약하는 λ-트랜지션 추상화 방법[1] 등이 소개되어 있다. 그림 1에서 볼 수 있듯이, 이러한 방법들은 크게 분석 상태 공간을 최소로 생성하는 기술과 이미 생성된 상태공간을 효과적으로 표현하거나 추상화 하는 기술로 나뉘어 볼 수 있다. 이 연구에서는 이들 중에서 이미 생성된 FSM 시스템 모델의 추상화에 가장 널리 사용되고 있는 내부 트랜지션 추상화 알고리즘의 문제점을 보완한다.

기존의 내부 트랜지션 추상화 방법[1]은 시스템의 모든 상태를 나열한 후에 추상화 과정을 수행하여야 하므로 시스템 규모가 커지면 상태 폭발이 발생하여 적용하지 못하는 단점이 있다. 이러한 기존의 추상화 알고리즘의 단점을 보완하기 위해 모든 시스템 상태를 나열하지 않고 필요한 시스템의 상태만을 부분적으로 생성하여 추상화할 수 있는 효과적인 방법을 제안한다. 먼저 유한상태머신의 정보를 모두 표현하면서 병행적 특성을 효율적으로 나타낼 수 있는 slice 모델 개념을 제안하고 이를 기반으로 부분적으로 분석 공간을 생성하여 상태 공간을 축소하는 추상화 방법을 제공한다. 향상된 추상화 방법은 기존의 추상화 방법으로는 상태폭발로 인해 적용하지 못하는 대규모의 시스템에도 적용 가능하므로 복잡한 시스템의 분석에 효과적으로 활용될 수 있다. 그림 1의 아래쪽 부분은 새롭게 제안하는 추상화 방법과 기존 관련연구와의 연관관계를 비교하여 설명하고 있다. Slice 모델 개념은 시스템 모델의 한 유형으로 전체 분석 공간 생성 이전에 적용되는, 독립적인 기술이므로 기존 연구들과 함께 적용될 수 있다. 이 논문에서 제안한 기법을 이용하면, 시스템 규모가 큰 시스템일지라도 slice 모델을 만든 다음, 분석과 무관한 트랜지션들을 축약한 후, 가능한한 분석 공간을 최소로 축약한 다음, 기존의 다양한 분석을 수행할 수 있다.

논문의 구성은 다음과 같다. 먼저 2장에서는 유한상태머신의 일종인 Petri net 모델의 도달성 그래프를 예제로 유한상태머신의 한계에서 대해 다룬다. 3장에서는 시스템 상태변수에 따라 분할하여 재구성한 slice 모델에 대해 정의하고 Petri net 모델을 slice 모델로 변환하는 규칙을 기술한다. 4장에서는 slice 모델을 이용한 트랜지션 축약 알고리즘으로 λ-루프와 λ-트랜지션 제거 알고리즘을 설명한다. 5장에서는 기존의 λ-트랜지션 축약 알고리즘과 slice 모델 기반의 향상된 λ-트랜지션 축약 알고리즘의 수행시간을 비교한다. 마지막으로 6장에서는 결론을 맺고 향후 연구방향을 소개한다.

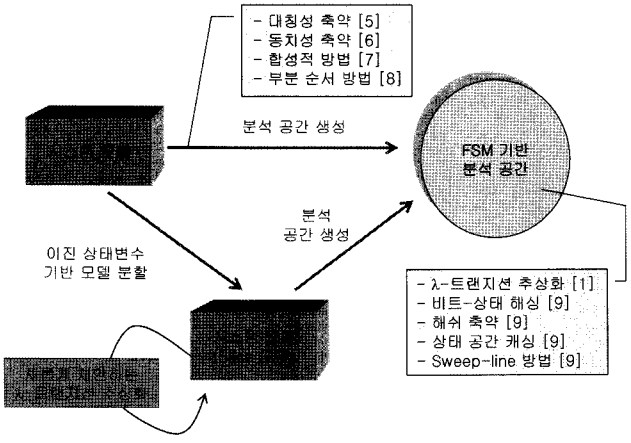


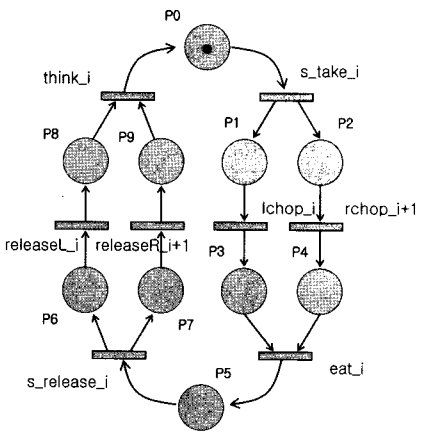
그림 1 Slice 모델 개념과 기존 연구와의 연관관계

2. 유한상태머신의 문제점

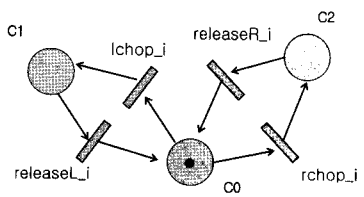
단순한 구조를 지닌 유한상태머신(Finite State Machine: FSM)은 시스템 분석이 간단하므로 비교적 적은 규모의 시스템 모델링 및 분석에 활용되고 있지만 시스템이 복잡해지면 모델링 및 분석이 어려워지는 단점이 있다. 복잡한 시스템의 경우는 주로 Statecharts[2], Petri nets[3] Labeled Transition Systems(LTS)[4] 등의 표현력이 뛰어난 정형적 방법들을 이용한다. Statecharts는 계층적인 측면, 병행적인 측면 등에서 FSM을 확장한 정형적 방법이며, LTS은 컴포넌트 단위로 FSM 모델을 기술하고 레이블을 이용하여 컴포넌트 간의 의존성을 표현하며, Petri nets은 병행적인 특성을 지닌 시스템을 표현하는데 주로 이용된다. 이러한 정형적 방법들의 분석에서는 먼저 시스템 모델을 FSM으로 변환한 다음, 교착상태 검사, 수행가능하지 않는 트랜지션 검사 등의 기존 FSM의 분석방법을 적용한다.

Petri nets으로 생각하는 철학자 문제를 모델링하고 FSM으로 변환하여 교착상태를 분석하는 경우를 예로 살펴보자. 그림 2(a)는 철학자의 행위를 나타내고 있다. Petri nets 모델에서 원으로 표현된 노드는 플레이스를 나타내고 직사각형 노드는 트랜지션을 나타낸다. Pi라는 이름의 플레이스는 토큰을 가지며 트랜지션은 입력되는 아크에 연결된 모든 플레이스에 토큰이 있어야 수행가능하다.

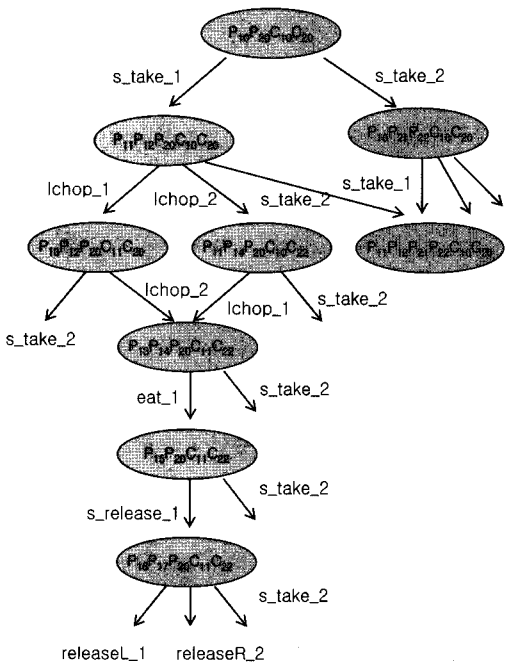
그림 2(a)에서는 s_take_i 트랜지션이 수행가능하다. 트랜지션이 수행되면 이전 플레이스에 있는 토큰은 사라지고 이후 플레이스에 토큰이 생성된다. 트랜지션 s_take_i가 수행되면 P0의 토큰은 사라지고 P1과 P2에 새로운 토큰이 하나씩 생성된다. 철학자가 좌우 젓가락을 하나씩 집거나 내려놓는 행위는 병행적으로 일어날 수 있으므로 병행적인 트랜지션으로 표현하고 있다. 그림 2(b)는 젓가락의 행위를 모델링하고 있다. 젓가락 모델의 플레이스는 Ci로 표현하고 있다. 그림 2(c)는 2명의 철학자가 있는 모델의 도달성 그래프의 일부를 나타낸 것이다. 도달성 그래프는 Petri nets 모델을 분석하기 위해 FSM 모델로 변환한 것이다. 도달성 그래프에서 상태는 토큰을 가지고 있는 플레이스들을 나타낸다. 그림에서 P_{ij}의 의미는 i번째 철학자 모델의 j번째 플레이스를 의미하며 C_{ij}의 의미는 i번째 젓가락 모델의 j번째 플레이스를 의미한다. 즉, 첫번째 상태인 "$P_{10}P_{20}C_{10}C_{20}$"는 1번 철학자의 P0, 2번 철학자의 P0, 1번 젓가락의 C0, 2번 젓가락의 C0에 토큰이 있음을 나타낸다. 그림 2(c)는 일부만 나타낸 것이며 실제로는 2 명의 철학자가 있는 모델의 도달성 그래프는 57개의 상태와 112개의 트랜지션으로 구성된다.



(a) i번째 철학자 모델



(b) i번째 젓가락 모델



(c) 철학자 2명인 경우의 도달성 그래프의 일부

그림 2 Petri net 모델과 도달성 그래프

하지만 Petri net 모델에 병행성이 포함되어 있고 규모가 커지는 경우에는 도달성 그래프의 상태의 수가 엄청나게 증가하는 상태폭발이 발생하므로 도달성 그래프 상에서의 분석은 불가능해진다. 예를 들어, 위의 철학자 모델에서 철학자의 수가 9명이 되면, 도달성 그래프의 상태의 수는 약 1억 개 정도가 되고 트랜지션의 수도 약 10억 개 정도가 되므로 PC급의 컴퓨팅 환경에서는 분석이 불가능하다. 병행적인 모델을 FSM 모델로 변환할 때, 상태의 수가 급격하게 증가하는 이유는 FSM 모델에서는 병행적인 순서관계를 나타내지 못하고 항상 전체순서관계로 표현하여야 한다. 예를들어, 그림 2(a)의 lchop_i와 lchop_i+1 트랜지션은 병행적으로 수행되지만, 그림 2(c)의 FSM 모델에서는 lchop_1 → lchop_2와 lchop_2 → lchop_1의 전체순서관계로 표현된다. 즉, n개의 병행적 이벤트가 있으면 FSM에서는 n! 만큼의 전체순서관계로 표현하여야 함으로 상태의 수가 급격하게 증가한다. 일부 Petri nets의 분석 방법에서는 FSM의 병행적 특성을 반영하지 못하는 단점 때문에 곧바로 도달성 그래프를 생성하지 않고 Petri nets의 병행적인 특성을 분할하여 Petri nets Slice로 분할하여 계층적으로 도달성 그래프를 생성 및 합성하는 연구[10]도 있다.

이 연구에서는 이러한 FSM의 병행적 특성을 제대로 나타내지 못하는 단점으로 인해 야기되는 상태 나열의 문제점을 극복하기 위해 시스템의 병행성을 전체순서관계로 나열하지 않고 그대로 표현하는 방법을 제시하며 이러한 모델을 기반으로 시스템 모델을 축약하는 방법을 제공한다.

3. 상태변수 기반의 Slice 모델

3.1 Slice 모델의 정의

FSM 모델에서는 시스템의 특정 순간의 상황을 상태로 나타내어 나열하고 상태간의 전이를 트랜지션으로 표현한다. 특히, 시스템의 상태는 시스템 상태변수의 특정 값들의 조합으로 표현된다. FSM 모델에서는 시스템 상태를 중심으로 기술되며 각각 상태변수에는 별로 관심을 두지 않는다. Slice 모델은 시스템 상태 중심의 FSM 모델과 달리 상태변수 관점에서 모델을 재구성한 것이다. 각각의 상태 변수 관점에서 트랜지션들이 어떻게 수행되는지를 세분화하여 나타내고 있다. 그림 3(a)는 on/off를 나타내는 v0 상태변수와 error의 유무를 나타내는 v1 상태변수가 있는 시스템의 행위를 3개의 상태를 가지는 FSM 모델로 나타내고 있다. 그림 3(b)는 각각의 상태 변수 v0, v1 관점에서 각각의 트랜지션들이 어떻게 수행되는지를 나타내고 있다. 트랜지션 on은 v0 상태변수를 0에서 1로 전이시키며, off와 error 트랜지션은 1에서 0으로 전이시킨다.

Slice 모델에서는 0과 1의 상태 값만 가지는 이진 상태변수로 구성된다고 가정하며 아래와 같이 정의된다.

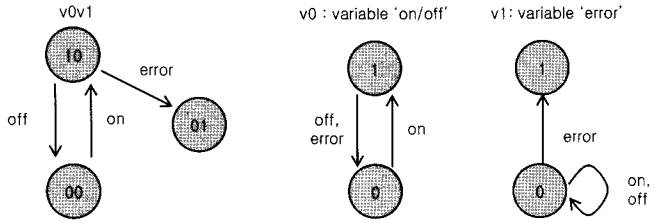
정의 1. Slice 모델

'0' 또는 '1'의 상태와 (00, 01, 10, 11) 유형의 트랜지션으로 정의되는 유한상태머신을 slice 모델이라 한다. 여기에서 '01' 트랜지션의 의미는 '0' 상태에서 '1' 상태로의 트랜지션을 나타낸다.

시스템의 행위는 일반적으로 이진 상태변수의 수만큼의 slice 모델의 집합으로 기술된다. 그리고 slice 모델간의 의존성은 트랜지션의 레이블로 표현된다. Slice 모델의 트랜지션은 병행적 합성(parallel composition) 개념[4]에 따라 수행되며 서로 다른 Slice 모델에 독립적으로 존재하는 지역 트랜지션들은 병행적으로 수행되며, 여러 Slice 모델에 공유된 트랜지션은 동기화하여 수행된다. 이와 같이 Slice 모델은 FSM 모델을 상태변수별로 독립적으로 재구성함으로써 상태변수간의 병행적인 특성은 그대로 나타낼 수 있다.

3.2 Petri net 모델에서의 slice 모델 생성

Petri net 모델을 slice 모델로 변환하기 위해서는 Petri net 모델을 0과 1 상태만을 가지는 이진변수들로 나타내어야 한다. Petri net 모델을 slice 모델로 변환하는 방법은 크게 Petri net 모델을 곧바로 바꾸는 방법과 Petri net 모델을 도달성 그래프로 변환한 다음에 slice 모델로 다시 변환하는 방법이 있다. 도달성 그래프로 변환한 후에 다시 변환하려면 모든 시스템 상태를 비효율적으로 나열하여야 함으로 이 연구에서는 Petri net 모



(a) FSM 모델 (b) Slice 모델

그림 3 FSM 모델의 Slice 모델 변환

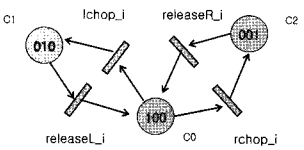
델에서 곧바로 slice 모델로 변환하도록 한다. Petri net 모델의 각각의 플레이스들은 하나의 이진 상태변수로 볼 수 있다. 플레이스에 토큰이 존재하면 1인 상태, 그렇지 않으면 0인 상태로 본다. Petri net 모델에 있는 각 플레이스들을 이진 상태변수로 간주하며 각 상태변수 별로 slice 모델을 정의하고 각 트랜지션들에 대해 정보를 각 slice 모델에 기록한다. Petri net 모델을 slice 모델로 변환하는 단계는 아래와 같다.

- 단계 1: Petri net 모델에 있는 각 플레이스를 0과 1의 상태만을 가지는 slice 모델로 정의한다.
- 단계 2: Petri net 모델의 각 트랜지션의 수행에 따른 토큰의 변화를 각 slice 모델에 반영한다. 하나의 트랜지션은 입력 플레이스들을 1 상태에서 0 상태로 변환하며 출력 플레이스들을 0 상태에서 1 상태로 변환한다. 이러한 상태변화에 따라 트랜지션의 이름을 해당 slice 모델의 '00', '01', '10', '11' 영역에 기록한다.

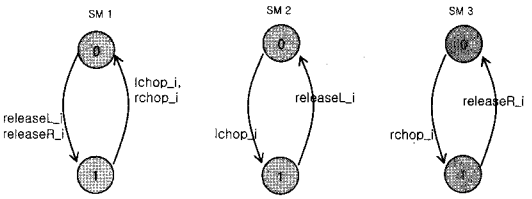
그림 4는 그림 2(b)에 나타난 젓가락 모델을 slice 모델로 변환하는 예를 보여준다. Petri net 모델에 3개의 플레이스가 있으므로 상태변수 3개로 표현된다. 즉, 상태변수 S1, S2, S3로 표현되어 C1은 100, C2는 010, C3는 001로 표현된다. 각 상태변수 별로 하나의 slice 모델이 생성되므로 총 3개로 정의된다. SM1은 상태변수 S1에 대응하는 Slice 모델로 S1만 고려하므로 C1과 C2는 0의 상태로 바뀌고 C0는 1의 상태로 변환된 것으로 볼 수 있다. SM2는 상태변수 S2에 대한 것이므로 C0와 C2가 0의 상태로 바뀌고 C1이 1의 상태로 변환된 것이다. 그림 4의 slice 모델에서 모든 트랜지션들은 공유 트랜지션이다. 앞서 언급한 공유 트랜지션의 수행조건을 구체적인 예로 다시 살펴보자. 트랜지션 releaseL_i는 SM1과 SM2 모델에 동시에 나타나는 공유 트랜지션으로 수행조건은 SM1은 0 상태, SM2는 1 상태일 때만 수행될 수 있다. SM3에서는 releaseL_i 트랜지션이 나타나지 않으므로 트랜지션 수행과는 무관하다.

보조정리 1. Petri net 모델의 각 트랜지션의 입출력 정보는 위의 변환 규칙에 따라 변환된 slice 모델에서도 그대로 유지된다.

증명. 변환 규칙에 따라 Petri nets의 각 플레이스는 고유의 상태변수가 부여되므로 각 플레이스는 해당 상태변수의 slice 모델로 매핑될 수 있다. 예를들어, 그림 4에서는 C0는 SM1, C1은 SM2, C2는 SM3로 매핑된다. 그리고 각 트랜지션의 입력 아크 정보는 입력 플레이스에 해당하는 slice 모델에 '10' 트랜지션으로 변환되며 출력 아크 정보는 출력 플레이스의 slice 모델에 '01' 트랜지션으로 일대일로 변환된다. 예를들어, C0에서 C1으로 가는 lchop_i 트랜지션의 경우에, C0에 해당하는 SM1에서는 '10'에, C1에 해당하는 SM2에서는 '01'로



(a) i번째 젓가락 모델



(b) Slice Model

그림 4 Petri net 모델을 Slice 모델로 변환한 예

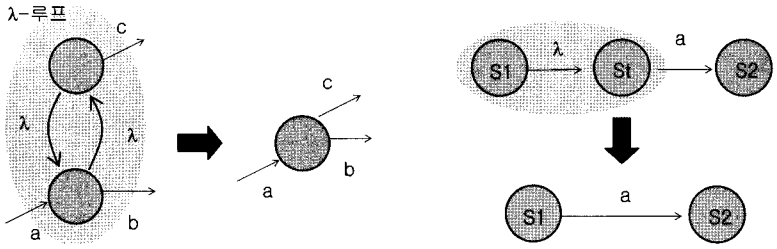
변환된다. 이와 같이 각 트랜지션의 입출력 아크 정보는 변환규칙에 따라 slice 모델에서도 그대로 유지됨을 알 수 있다. □

정리 1. 위의 변환 규칙에 따라 변환된 slice 모델과 원래의 Petri net 모델은 동일한 시스템 행위를 가진다.

증명. 두 모델의 행위 일치성을 보이기 위해 두 모델의 트랜지션 정보가 동일하고 트랜지션의 수행에 따라 동일한 상태변화가 일어남을 보인다. 보조정리 1에서 두 모델이 동일한 트랜지션 정보를 가짐을 보였으므로 여기에서는 트랜지션의 수행에 따라 동일한 상태 변화가 일어남을 보인다. Petri net에서 트랜지션이 수행되기 위해서는 입력 플레이스에 토큰이 존재하여야 하며 트랜지션이 수행되면 입력 플레이스의 토큰이 사라지고 출력 플레이스에 토큰이 생성된다. Slice 모델에서는 하나의 트랜지션이 수행되기 위해서는 각 slice 모델에서 공유되는 트랜지션 레이블이 동시에 수행되므로 입력 플레이스에 해당하는 10 트랜지션과 출력 플레이스 해당하는 01 트랜지션이 동시에 수행된다. 이는 결과적으로 입력 플레이스에 해당하는 slice 모델은 0의 상태(즉, 토큰이 없는 상태)로, 출력 플레이스에 해당하는 slice 모델은 1의 상태(즉, 토큰이 생성된 상태)로 변환됨을 알 수 있다. □

4. Slice 모델에서의 트랜지션 축약 알고리즘

FSM 모델의 트랜지션 축약 알고리즘은 시스템 모델의 내부 정보를 나타내는 트랜지션을 추상화하는 방법으로 모델의 분석 공간을 축소하므로 향후 효율적인 분석을 수행할 수 있게 한다. 트랜지션 축약 알고리즘은 λ-축약 방법이라고도 하며 크게 λ-루프 축약 단계와 독립적인 λ-트랜지션 축약 단계로 나뉘어진다[1]. 그림 5



(a) λ-루프 축약 (b) 독립적인 λ-트랜지션 축약

그림 5 λ-루프와 λ-트랜지션 축약의 예

는 λ-루프 축약 예와 독립적인 λ-트랜지션 축약 예를 보여준다.

하지만 기존의 트랜지션 축약 알고리즘은 FSM 모델 상에서 동작하므로 FSM 모델을 생성하기 어렵거나 방대한 FSM 모델인 경우는 적용하는데 어려움이 있다. Slice 모델에서의 트랜지션 축약 방법은 전체 FSM 모델을 모두 생성하는 기존 방법과 달리 부분적으로 필요한 분석 공간만을 동적으로 생성하여 수행하므로 효율적으로 수행될 수 있다.

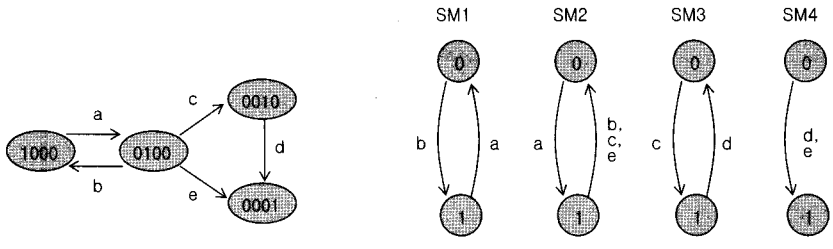
4.1 Slice 모델의 λ-루프 축약 알고리즘

전체 상태 공간을 검색하여 λ-루프를 찾는 기존 알고리즘과 달리 Slice 모델의 루프 검사 기법은 상태변수에 나타나는 루프의 특성을 이용하여 검색한다. 루프는 기본적으로 루프를 구성하는 상태변수들을 모두 초기상태로 되돌리므로 각 연관된 각각의 slice 모델에서도 반드시 루프가 존재하게 된다.

그림 6은 이러한 루프의 특성을 보여준다. 그림 6(a)는 루프를 포함하는 FSM 모델을 보여주며, 그림 6(b)는 FSM 모델을 4개의 slice 모델로 변환한 예를 보여준다. FSM 모델에는 트랜지션 a와 b로 구성된 루프가 있으며, 트랜지션 a와 b에 연관된 slice 모델인 SM1과 SM2에서 각각 해당 트랜지션으로 구성된 루프가 존재한다.

이러한 루프 특성을 이용하여 slice 모델에서는 λ-루프를 검색하여 추상화한다. Slice 모델에서의 λ-루프 제거 알고리즘은 아래와 같이 의사 코드로 나타낼 수 있다. 먼저 λ-트랜지션으로 구성될 수 있는 모든 조합을 구한 다음에 각 조합 항목에 대해, 연관된 slice 집합을 구성하고 이러한 slice 모델들이 루프 특성을 만족하는지를 검사한다. 만약 루프 특성을 만족하면 실제 루프가 구성되는지 검사하고 루프가 존재하는 경우에는 루프 추상화를 수행한다. 루프 추상화를 수행하면 λ-트랜지션의 집합이 변경되므로 전체 과정을 처음부터 반복하여 새로운 λ-루프가 있는지를 다시 검사한다. 이러한 반복과정은 λ-루프가 발견되지 않을 때까지 수행된다.

예를 들어, 그림 6(b)에 나타난 slice 모델에서 λ-루프를 검색하는 과정을 살펴보자. 만약 관심이 없는 트랜지션 집합이 {a, b}인 경우이면, 먼저 a와 b 트랜지션으로 이루어진 셀프 루프가 있는지를 검사한다. 트랜지션 a의 경우는 SM1과 SM2에 나타나 있지만 두 slice 모델에서 a만으로 이루어진 루프가 없으므로 트랜지션 a의 셀프 루프는 없다. 트랜지션 b의 셀프 루프도 마찬가지로 존재하지 않는다. 다음은 a와 b로 이루어진 루프를 검색한다. 트랜지션 a, b와 연관된 slice 모델은 SM1과 SM2이다. 두 slice 모델에서 각각 a와 b로 구성된 루프가 존재한다. 즉, a와 b로 구성된 λ-루프가 발견된 것이다. 또다른 예로 관심이 없는 트랜지션 집합을 {a, c, d}라고 가정해 보자. 연관된 slice 모델로는 SM1, SM2, SM3, SM4가 모두 해당된다. SM2와 SM3에서는 해당 트랜지션으로 구성된 루프가 존재하지만 SM1과



(a) FSM 모델의 루프 (b) Slice 모델의 루프

그림 6 트랜지션 루프의 특성

```
Name : lamda loop elimination
Input : slice set
Output : slice set without lamda loops
void loop_elimination(slice set)
{
   changed = true;
   while ( changed ) {
        changed = false;
        for ( each combination of the λ transitions set ) {
                consist of related slice sets;
                check the loop condition in the related slice sets;
                if ( there is a loop ) {
                        perform a loop abstraction;
                        changed = true;
                        break;
                }
        }
   }
}
```

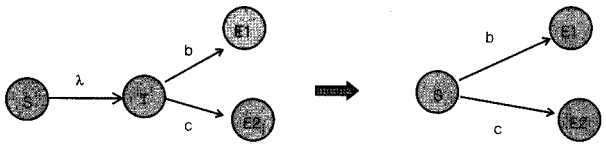
그림 7 λ-루프 축약 알고리즘의 의사코드

SM4에는 루프가 존재하지 않으므로 a, c, d로 구성된 루프는 존재하지 않는다.

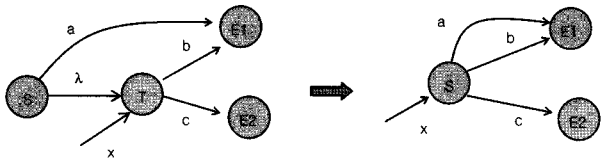
병행적으로 수행될 수 있는 부분에서 루프가 발생할 경우, FSM 모델에서는 병행적인 부분이 여러 군데에서 반복적으로 나타나므로 여러 곳에 나타나는 동일한 루프들을 모두 검색하여 제거하여야 하는 반면, slice 모델에서는 병행적인 부분에 대해서 한번만 루프를 검색하여 제거하면 된다. 또한 slice 모델에서의 λ-루프 제거 알고리즘에서는 전체 상태 공간을 검색할 필요 없이 루프에 연관된 slice 모델만 부분적으로 검색하므로 시스템의 전체 분석 영역의 규모에 크게 영향을 받지 않는다.

4.2 Slice 모델의 λ-트랜지션 축약 알고리즘

λ-루프를 제거한 후에 내부 상태 변화만을 일으키는 λ-트랜지션을 검색하여 축약한다. 모든 λ-트랜지션이 축약 대상인 것은 아니며 축약 전후의 외부 행위가 동일한 경우만 축약이 가능하다. 그림 8은 축약하더라도 외부 행위가 일치하는 독립적인 λ-트랜지션의 축약 예와 축약 전후의 외부 행위가 일치하지 않는 비독립적인 λ-트랜지션의 축약 예를 보여준다. λ-트랜지션 축약 알고리즘은 이러한 독립적인 λ-트랜지션들을 검색하여 제거한다.



(a) 독립적인 λ-트랜지션의 제거



(b) 비독립적인 λ-트랜지션 제거

그림 8 독립적인 λ-트랜지션의 제거 방법

FSM 모델에서 이러한 독립적인 트랜지션 검사를 효율적으로 수행하기 위해서는 상태의 입력 트랜지션 정보를 따로 저장하고 있어야 한다. 만약 입력 트랜지션 정보를 저장하고 있지 않으면 모든 트랜지션을 검색하여 특정 상태의 입력 트랜지션들을 확인하여야 하므로 수행시간이 많이 필요하다. 하지만 slice 모델에서는 순방향 수행 가능한 트랜지션 검사와 동일한 방법으로 역방향 수행 가능한 트랜지션들을 동적으로 생성할 수 있으므로 이러한 입력 트랜지션들을 보다 쉽게 검색할 수 있다. 그림 9는 Slice 모델에서의 독립적인 λ-트랜지션 축약 알고리즘을 의사코드 형식으로 보여준다.

```
Name : lamda transition elimination
Input : slice set
Output : slice set
void transition_elimination(slice set)
{
   for ( each λ transition in the slice set ) {
        find a set of slices related with the λ transition;
        if ( there is another backward firable transition )
              continue;
        else
              eliminate the λ transition;
   }
}
```

그림 9 독립적인 λ-트랜지션 축약 알고리즘의 의사코드

5. FSM과 Slice 모델의 트랜지션 축약 알고리즘의 비교 분석

병행적인 특성을 가진 시스템을 slice 모델로 표현하면 FSM과 달리 병행적인 특성을 그대로 나타낼 수 있으므로 효율적인 시스템 모델 기술이 가능하며 또한 slice 모델을 기반으로 효율적인 트랜지션 추상화 알고리즘을 제공할 수 있었다. 이 절에서는 시스템 모델의 크기 측면과 트랜지션 추상화 알고리즘의 효율성 측면에서 FSM 모델과 Slice 모델을 비교한다. 알고리즘들은 Intel 2.4Ghz CPU와 2 GB 메모리, Windows XP의 PC 환경에서 MS Visual C++에서 구현되었다.

표 1은 철학자 모델의 도달성 그래프에 대한 FSM 모델과 시스템 상태변수에 따라 재구성된 slice 모델의 크기를 비교하여 보여준다. 전체 상태 공간을 검사하는 경우를 보이기 위해 FSM과 Slice 모델에서 깊이-우선 검색 기법을 이용하여 교착상태 검사 수행 시간을 비교하였다. FSM 모델은 이미 만들어진 상태 모델에서 검사하므로 아주 짧은 시간 내에 수행되지만 철학자 수가 7 이상의 경우에는 FSM 모델이 없어 수행이 불가능하다. Slice 모델의 경우는 깊이-우선으로 전체 상태 공간을 동적으로 만들면서 교착상태를 검사하므로 FSM에 비해 좀더 많은 시간이 소요되지만 특정 순간에 일부의 분석 공간만을 생성하여 검색하므로 철학자 수가 많아지더라도 분석이 가능하다. 여기에서 수행시간이 아주 짧은 경우는 0 또는 0.015로 일관되지 않게 측정되어 이런 경우를 일관되게 표현하기 위해 0.015보다 적다는 의미로 "< 0.015"로 나타낸다.

표 1에서 알 수 있듯이 slice 모델은 병행적인 특성들을 합성하지 않고 그대로 표현하므로 모델의 크기는 철학자의 수가 증가하더라도 산술적으로 증가한다. 반면 도달성 그래프에서는 시스템에 있는 모든 병행적 특성들을 전체순서관계로 모든 경우의 수를 나열하여 나타내므로 철학자 수가 증가하면 시스템 상태의 수가 기하급수적으로 증가함을 알 수 있다.

표 2는 내부 트랜지션을 8개로 설정하여 철학자 수를 증가시킴에 따라 FSM 모델과 Slice 모델에서의 추상화 알고리즘의 수행 시간을 비교하고 있다. 추상화 알고리즘은 크게 λ-루프 제거와 λ-트랜지션 제거 알고리즘으로 나뉘어지므로 두 부분으로 나뉘어 수행시간을 비교한다.

표 2에서 알 수 있듯이, FSM의 λ-추상화 알고리즘은 철학자 수가 작을 때는 짧은 시간 내에 수행되지만 철학자 수가 많아지면 수행 시간이 급증함을 알 수 있다. 즉, 상태의 수가 많아지면, 모든 상태를 검색하여 λ-루프와 λ-트랜지션을 검사하여야 하기 때문에 수행 시

표 1 FSM 모델과 Slice 모델의 상태공간 및 교착상태 수행시간 비교 (단위:초)

철학자 수	FSM 모델 상태수 (아크수)	Slice 모델 상태수 (아크수)	깊이우선 교착상태 검사	
			FSM	Slice
2	57(112)	8(56)	< 0.015	< 0.015
3	446(1302)	12(84)	< 0.015	0.015
4	3429(13360)	16(112)	< 0.015	0.031
5	26255(127860)	20(140)	0.015	0.141
6	200952(1174356)	24(168)	0.125	5.156
7	-	28(196)	-	135.391
8	-	32(224)	-	2970.690

표 2 FSM과 Slice 모델의 추상화 알고리즘의 수행시간 비교 (단위:초)

철학자 수	λ-Loop 제거		λ-트랜지션 제거	
	FSM	Slice	FSM	Slice
2	< 0.015	< 0.015	< 0.015	< 0.015
3	< 0.015	< 0.015	< 0.015	< 0.015
4	< 0.015	< 0.015	< 0.015	< 0.015
5	0.187	< 0.015	0.015	< 0.015
6	2.297	< 0.015	0.218	< 0.015
7	-	< 0.015	-	< 0.015
8	-	< 0.015	-	< 0.015

표 3 내부 트랜지션의 수를 달리한 추상화 알고리즘의 수행시간 (단위:초)

트랜지션 수	λ-루프 제거 알고리즘	λ-트랜지션 제거 알고리즘
8	< 0.015	< 0.015
16	< 0.015	< 0.015
24	0.016	0.016
32	0.062	0.031
40	0.172	0.031
48	0.406	0.031
56	0.750	0.047
64	1.438	0.047
72	2.078	0.062

간이 급증한다. 반면, Slice 모델의 추상화 알고리즘은 시스템 규모보다는 내부 트랜지션의 수에 의존적이므로 철학자 수가 증가하더라도 수행 시간은 거의 동일하게 나타나고 있다.

표 3은 9 명의 철학자 모델에서 내부 트랜지션의 수를 증가시키면서 slice 모델의 추상화 알고리즘의 수행시간을 보여준다. 표 3에서 알 수 있듯이, 내부 트랜지션이 두 배씩 증가함에 따라 추상화 알고리즘의 수행시간도 산술적으로 증가됨을 알 수 있다.

Slice 모델의 추상화 알고리즘은 표 1과 표 2에 나타난 바와 같이 기존 FSM 모델 기반의 추상화 알고리즘에 비해 병행적인 특성을 효과적으로 나타낼 수 있으며 모델의 규모가 커지더라도 수행시간이 민감하게 반응하지 않음을 알 수 있다. 또한 표 3을 통해서 알 수 있듯이, slice 모델의 추상화 알고리즘은 추상화 적용 대상이 많아지더라도 수행시간이 산술적으로 증가함으로 내부 트랜지션의 규모에도 크게 민감하지 않음을 알 수 있다. 이상과 같이 slice 모델의 추상화 알고리즘은 기존의 추상화 알고리즘에 비해 시간 및 공간적으로 효율적으로 수행될 수 있음을 알 수 있다.

6. 결 론

본 논문에서는 시스템의 상태폭발을 최소화하는데 이용될 수 있는 향상된 트랜지션 추상화 알고리즘을 제공하였다. 제안된 트랜지션 추상화 알고리즘은 slice 모델 개념에 근간을 두고 동작되며 기존 트랜지션 추상화 알고리즘에 비해 수행 시간, 메모리 사용면에서 효율적이며 또한 시스템 규모가 커지더라도 크게 영향을 받지 않는 특성을 가지는 장점이 있다. 또한 이러한 축약 방법들은 기존의 다른 축약방법들과 병행적으로 사용될 수 있으며 시스템을 관심 있는 트랜지션들을 중심으로 추상화하여 시스템 모델을 축소할 수 있으므로 다양한 분석 기법에 활용될 수 있다.

현재 slice 모델과 트랜지션 추상화 알고리즘은 텍스트 기반으로 구현되어 있어 시스템 모델을 텍스트 형태로만 입력하여 사용하여야 함으로 입출력에 불편함이 많다. 이러한 불편함을 해결하기 위해서는 그래픽 기반의 모델링 도구와 연계되어 동작하여야 하며 또한 다양한 그래픽 기반 분석 도구와도 연계할 필요가 있다. 향후 연구방향으로는 이러한 그래픽 모델 도구와 다양한 분석 도구와의 연계방법을 모색하고 있다.

참 고 문 헌

[1] P. J. Denning, et al., *Machines, Languages, and Computation*, Prentice Hall, 1978.
[2] D. Harel, "Statcharts: A visual formalism for complex systems," Sci. Comput. Prog., Vol.8, pp. 231-274, 1987.
[3] W. Reisig, Petri Nets : An Introduction, Springer-Verlag, 1985.
[4] Robin Milner, *Communication and Concurrency*, Prentice Hall, 1989.
[5] E. Clarke, E. Emerson, S. Jha, and A. Sistla, "Symmetry reductions in model checking," in Proc. CAV, LNCS #1427, pp. 147-159, 1998.
[6] K. Jensen, "Condensed state spaces for symmetrical coloured Petri nets," Formal Meth. Syst. Design, Vol.9, pp. 7-40, Aug. 1996.
[7] W. J. Yeh, M. Young, "Compositional reachability analysis using process algebra," Proc. of ACM SIGSOFT, pp. 49-59, 1991.
[8] A. Valmari, "Stubborn sets for reduced state space generation," Proc. of Advanced in Petri nets, pp. 491-515, 1990.

[9] J. Billington, G. E. Gallasch, L. M. Kristensen, and T. Mailund, "Exploiting equivalence reduction and the sweep-line method for detecting terminal states," IEEE Trans. on Systems, Man, and Cybernetics - Part A: Systems and Humans, Vol. 34, No.1, Jan. 2004.

[10] 이우진, 차성덕, 권용래, 김홍남, "페트리 네트Slice를 이용한 페트리네트 모델의 합성적 분석", 정보과학회 논문지: 소프트웨어 및 응용, Vol.29, No.3, 2000년 3월

이 우 진

1992년 경북대학교 컴퓨터과학과 졸업(학사). 1994년 한국과학기술원 전산학과 졸업(공학석사). 1999년 한국과학기술원 전산학과 졸업(공학박사). 1999년~2002년 한국전자통신연구원 S/W공학연구부 선임연구원. 2002년~현재 경북대학교 전자전기컴퓨터학부 조교수. 관심분야는 임베디드 실시간 시스템 모델링 및 분석, Requirements Engineering, Petri nets, 웹 서비스 기술 등