기호적 상태 표현 방법을 이용한 병행적 시스템의 검증*

김한상○, 최진영
고려대학교 컴퓨터학과 정형기법연구실

A Formal Verification of Concurrent System using Symbolic State Method

Hahnsang Kim○, Jin-Young Choi
Formal Methods Lab. Dept. of Comp. Sci. & Eng. Korea Univ.

요 약

concurrent reactive system은 일반적으로 매우 많은 다른 행동들(different behaviors)을 보여 주기 때문에 설계하기가 어렵다. 또한 한 시스템에서의 구성 요소들 간에 상호작용을 함으로 전체 시스템의 상태를 나타낸다면 전체 상태는 각 구성 요소의 상태들의 곱으로 나타내지므로 복잡도는 지수의 곱으로 나타내 질 것이다. 따라서 단순한 테스트로는 시스템 특성의 정확성을 검증하기 위해 모든 상태를 확인할 수 없다.

본 논문에서는 concurrent system의 고전적 문제인 dining philosophers problem을 상징적 기호 표현 방법을 이용해서 그 시스템의 정확성을 검증하고 또한 상태 공간 폭발 문제-상태 공간의 크기가 시스템에서 구성요소의 복잡도와 수가 동시에 지수적으로 증가하는 문제-를 어떻게 해결할 수 있는가를 제시한다.

1. 서 론

concurrent system은 병행적으로 상호 작용을 하는 구성 요소들(elements)로 구성되어 있다. 각 구성 요소는 프로세스의 환경과 끊임없이 상호 작용하는 반응 시스템(reactive system)이라고 볼 수 있다. 이런 반응 시스템은 데이터의 처리에 중점을 두기보다는 프로세스들의 제어에 중점을 둔다. 이런 concurrent reactive systems에는 컴퓨터 네트워크(computer networks), 비동기 회로(asynchronous circuits), 운영체제(operating systems), 비행기 제어 시스템(flight-control systems), 공장 제어 시스템(plant-controller systems) 등을 들 수 있다.

concurrent reactive system은 일반적으로 매우 많은 다른 행동들(different behaviors)을 보여 주기 때문에 설계하기가 어렵다. 한 시스템의 병행적 구성요소들 간의 모든 가능한 상호작용을 생각해 보면 지수적으로 상태가 증가함을 알 수 있다. 따라서 단순한 시뮬레이션이나 테스트로는 시스템의 정확한 동작을 보장할 수 없다. 안전성이 보장되어야 하는 비행기 제어 시스템이나 경제적 손실 유무에 민감한 전화 교환기와 같은 제어 시스템에서는 단 한 번, 한가지의 오류도 용납이 되어서는 안 된다.

검증은 concurrent reactive system의 설계상의 정확성을 확인시켜주는 수단을 제공한다. 검증이란 한 시스템의 명세가 그 시스템이 요구하는 특성을 올바르게 수행하는지를 체크하는 것을 말한다. 한 시스템이 원하는 특성을 따르는가를 확인하기 위해서는, 이 시스템의 구성 요소가 주어진 특성에 부합되는지 그 시스템의 모든 가능한 행동들(all possible behaviors)을 체크해야 한다. 유한 상태 concurrent reactive system을 분석하고 검증하는 가장 좋은 방법이 모든 상태 공간을 탐색하는 방법이다. 그러나 한 시스템의 가능한 상태는 지수적으로 증가한다.

상태 공간 폭발 문제-상태 공간의 크기가 시스템에서 구성 요소의 복잡도와 수가 동시에 지수적으로 증가하는 문제-를 해결하면서 한 시스템의 특성의 정확성에 대한 검증을 concurrent reactive system의 고전적인 문제인 dining philosophers problem을 예로 보이고자 한다.

본 논문의 구성을 살펴보면, 2장에서는 concurrent processing에 대한 여러 가지 동기화 문제들 중에 대표적인, 고전적인 문제인 Dining Philosophers Problem을 바탕으로 발생되는 문제점과 접근 방법을 설명하고, 3장에서는 dining philosophers problem을 기호적 상태 표현 방법을 이용해서 그 시스템의 정확성을 검증하고 또한 상태 공간 폭발 문제를 어떻게 해결할 수 있는가를 제시한 후, 4장에서 결론을 맺겠다.

2. Dining Philosophers Problem

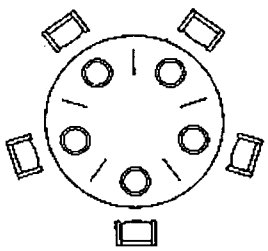

그림 1 : dining philosophers 상황

*본 연구는 1997년도 특정기초연구지원사업(97-01-00-09-01-3)으로 연구되었음.

concurrency control에서 항상 여러 가지 동기화 문제가 발생한다. 그 동기화의 고전적인 문제들 중의 하나가 Dining Philosophers Problem이다[1, 2].

5명의 철학자들이 생각하면서 식사를 하고 있다. 철학자들은 원형 테이블에서 5개의 의자에 앉아 있다. 테이블 중앙엔 밥그릇이 있고 5개의 젓가락이 있다. 철학자가 생각할 때는 다른 동료들과 이야기하지 않는다.

때때로 철학자들은 배가 고프면 가장 가까이 있는 두 개의 젓가락을 잡는다(젓가락은 왼쪽 철학자 그리고 오른쪽 철학자 사이에 있다). 이와 같은 상황에서 다음과 같은 규칙에 따라서 철학자들은 식사한다. 철학자는 한번에 한 개의 젓가락만 집을 수 있다. 그러나 옆 사람의 손에 들어간 젓가락을 집을 수는 없다.

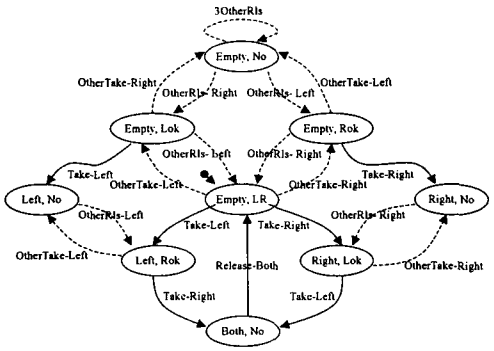

그림 2 : 한 프로세스(철학자)의 상태도

빼고픈 철학자가 동시에 젓가락 두 개를 잡으면, 젓가락을 놓지 않고 식사를 한다. 식사를 마치면, 젓가락 두 개를 모두 놓고 다시 생각을 시작한다. 한 철학자의 입장에서 위의 규칙에 따라 상태가 변하는 상태도를 그릴 수 있다[그림 2].

그림 2에 대한 내용을 살펴보면 다음과 같다. 철학자를 하나의 프로세스라고 생각하면, 프로세스가 가질 수 있는 상태는 표 1과 같고, 또한 프로세스가 행할 수 있는 동작은 표 2와 같다. 한 상태에서 다른 상태로의 전이는 화살표로 나타내고 각 화살표에 프로세스의 동작이 할당되어 있다. 화살표의 실선은 자신의 프로세스가 능동적으로 동작할 상황에서의 전이를 나타내고, 화살표의 점선은 이웃하는 프로세스(이웃하는 철학자)가 동작했을 경우에 자신의 프로세스가 영향을 받아 수동적으로 전이되는 것을 나타낸다.

일련의 과정을 설명하면, 한 프로세스의 초기 상태는 {Empty, LR}이다. 이것의 의미는 현재 손에는 아무 것도 잡고 있지 않고(Empty), 양쪽 젓가락을 잡을 수 있는 상태(LR)을 나타낸다. 여기서 Take-Left를 하면 왼쪽 젓가락을 잡고있는 상태({Left, Rok})가 되고, 오른쪽에 이웃하는 프로세스가 왼쪽 젓가락을 잡는 경우(OhterTake-Left 동작 실행) 자신의 프로세스는 왼쪽의 젓가락을 잡고 있지만 오른쪽은 잡을 수 없는 상태({Left, No})가 된다.

이와 같은 상태({Left, No})가 되었을 경우, 이웃하는 프로세스가 자신의 오른쪽 젓가락을 놓는 동작(OtherRls-Left)을 해주지 않으면 자신의 능동적 동작으로 {Left, No}상태를 빠져 나올 수 없음을 상태도를 통해서 쉽게 확인할 수 있다. 이와 같은 상태는 모든 프로세스가 왼쪽의 젓가락을 잡고 오른쪽 젓가락이 놓여지기를 기다리는 상태로 전형적인 교착 상태이다.

상태	의미
Empty	프로세스가 어느 쪽에도 젓가락을 들고 있지 않는 상태
Left	프로세스가 왼손에 젓가락을 들고 있는 상태
Right	프로세스가 오른손에 젓가락을 들고 있는 상태
Both	프로세스가 양손에 젓가락을 들고 있는 상태

표 1 : 한 프로세스가 가질 수 있는 상태

동작	의미
Take-Right	자신의 프로세스가 오른쪽 젓가락을 잡는 동작
Take-Left	자신의 프로세스가 왼쪽 젓가락을 잡는 동작
Release-Both	자신의 프로세스가 양손에 갖고 있는 젓가락을 동시에 놓는 동작
OtherTake-Left	오른쪽에 이웃하는 프로세스가 왼쪽 젓가락을 잡는 동작(자신의 프로세스는 오른쪽의 젓가락을 쓸 수 없다.)
OtherTake-Right	왼쪽에 이웃하는 프로세스가 오른쪽 젓가락을 잡는 동작(자신의 프로세스는 왼쪽의 젓가락을 쓸 수 없다.)
OtherRls-Left	오른쪽에 이웃하는 프로세스가 왼쪽 젓가락을 놓는 동작(자신의 프로세스는 오른쪽의 젓가락을 쓸 수 있다.)
OtherRls-Right	왼쪽에 이웃하는 프로세스가 오른쪽 젓가락을 놓는 동작(자신의 프로세스는 왼쪽의 젓가락을 쓸 수 있다.)

표 2 : 프로세스의 동작

교착 상태를 해결하는 방법으로는 비대칭적 방법(Asymmetric solution)과 대칭적 방법(Symmetric solution)이 있다. 본 논문에서는 대칭적 방법을 이용해 dining philosophers problem을 해결하는 알고리즘이 프로세스(철학자)의 수에 무관하게 교착 상태를 해결할 수 있는가의 방법을 제시하고자 한다[3].

3. 기호적 상태 표현 방법을 이용한 문제 해결

dining philosophers problem을 살펴보면 다음과 같은 특성이 있음을 알 수 있다.

관찰 1. 각각의 프로세스(철학자)는 대칭적(symmetric)이다. 따라서 시스템의 정확성의 체크에 영향을 주지 않고 바꿀 수 있다.
관찰 2. 이 시스템은 동종(homogeneous)의 요소로 구성되어 있어서 각 프로세스의 행동(behavior)은 같은 유한 상태 기계(finite state machine)로 나타낼 수 있다.
관찰 3. 모든 철학자들 중 적어도 한 철학자 이상이 두 젓가락을 잡고 식사할 수 있다.

위의 관찰을 통해서 이 시스템에서 특정한 상태를 갖는 프로세스의 수가 어떤 한 시스템의 상태가 허용되느냐 안되느냐를 판단하는데 중요한 역할을 함을 알 수 있다. 예를 들면, 한 시점에서 {Both, No}의 상태를 갖는 프로세스(철학자)는 오직 하나만 존재해야 한다. {Left, No}의 상태를 갖는 프로세스는 하나 이상 즉 다수 존재할 수 있다. 이론상 시스템의 정확성에 아무런 영향을 주지 않고, 프로세스(철학자)의 수는 무한정 생길 수가 있는 것이다. 따라서 모든 경우에 있어서 특정한 상태를 갖는 프로세스의 수가 정확히 몇 개인지는 중요치가 않다. 중요한 것은 주어진 상태에서 프로세스의 수가 없는지, 하나인지 아니면 다수 존재하는 가이다. 이러한 상태를 기호적으로 나타내기 위해 repetition constructors를 사용한다[4].

정의 : repetition constructors
(1) 0은 어떤 상태를 갖는 프로세스가 존재하지 않는다.
(2) 1은 어떤 상태를 갖는 프로세스가 오직 하나 존재한다.
(3) +는 어떤 상태를 갖는 프로세스가 1 혹은 다수 존재한다.
(4) *는 어떤 상태를 갖는 프로세스가 0, 1, 혹은 다수 존재한다.

따라서, 각 constructors간의 순서는 1<+<* 와 0<* 임을 알 수 있다. 결론적으로 dining philosophers problem에서 반드시 만족해야할 특성들은

1. 반드시 한번 이상의 {Both, No}상태를 갖는다.
2. ({Left, No}, {Right, No})가 존재해서는 안된다. 다시 말하면 {Left, No}상태와 {Right, No}상태가 동시에 만족될 수 없다.

1의 특성은 모든 철학자들 중 적어도 한 철학자 이상이 두 젓가락을 잡고 식사할 수 있다는 의미이고 2의 특성은 각각의 철학자가 동시에 왼쪽의 젓가락을 잡고 있는 상태 혹은 오른쪽의 젓가락을 잡고 있는 상태가 발생하지 말아야 한다는 의미이다. 즉 교착상태(deadlock)를 체크하는 조건이다. 그림 2의 문제가 1과 2의 특성을 만족하는지 알아보자.

1. ({**Empty, LR**}* ; *Take-Left*)
2. ({**Empty, LR**}* ; *Take-Left&OtherTake-Right*)
3. ({**Left, Rok**}, {**Empty, Rok**}, {Empty, LR}*)
4. ({**Left, Rok**}, {Empty, Rok}, {Empty, LR}* ; *Take-Right&OtherTake-left*)
5. ({**Both, No**}, {**Empty, Lok**}, {Empty, Rok}, {Empty, LR}*)
6. ({Both, No}, {**Empty, Lok**}, {Empty, Rok}, {Empty, LR}* ; *Take-Left&OtherTake-Right*)
7. ({Both, No}, {**Left, No**}, {**Empty, Rok**}, {Empty, Rok}, {Empty, LR}*)
8. ({Both, No}, {Left, No}, {**Empty, Rok**}*, {Empty, LR}*)
9. ({Both, No}, {Left, No}, {**Empty, Rok**}*, {Empty, LR}* ; *Take-Right&OtherTake-Left*)
10. ({Both, No}, {Left, No}, {**Right, No**}, {**Empty, Lok**}, {Empty, Rok}*, {Empty, LR}*)

10번째에서 {Left, No}와 {Right, No}가 동시에 발생했음을 알 수 있다. 다시 말해서 교착상태가 되었음을 알 수 있다. 또한 7번째에서 8번째를 보면 constructor '+'를 이용해서 상태 수가 줄어들었음을 알 수 있다.

concurrent system의 특성을 검증하는데 시스템의 모든 가능한 상태를 확인하기 위해서는 상태공간 폭발 문제를 우선적으로 해결해야 한다. 이와 같은 상태를 상태 기호로 나타냄으로써 상태공간을 엄청나게 줄일 수 있고, 시스템이 만족해야할 특성을 쉽게 확인, 검증할 수 있다. 또한 이러한 프로세스의 상태를 기호화함으로써 상태 공간을 줄일 수 있다.

4. 결 론

concurrent system에서 고전적 문제인 dining philosophers problem의 효율적 검증 방법을 제시하고자 하는 것이 이 논문의 방향이다. 이 방법은 concurrent system의 dining philosophers problem의 대칭성(symmetry), 동질성(homogeneity)을 이용해서 같은 상태를 갖는 프로세스를 묶는 상징 기호적 방법이다. 이런 기호적 방법은 concurrent system 특성의 정확성을 검증하는 데 효율적이며, 동일한 상태를 합침으로써 상태 공간의 크기을 급격하게 줄일 수 있다는 것이 가장 큰 장점이다.

5. 참고문헌

[1] Abraham Silberscharz and Peter B. Galvin, "Operating System Concepts fourth edition", Addison Wesley.
[2] E. W. Dijkstra, "Cooperating Sequential processes," Technical Report EWD-123, Technological University, Eindhoven, the Netherlands(1965); pages 43-112.
[3] Patrice Godefroid, "Partial-Order methods for the Verification of Concurrent Systems *An Approach to the State-Explosion Problem*", PH.D Dissertation, University De Liege, 1994-1995.
[4] F. Pong and M. Dubois, "Formal Verification of Complex Coherence Protocols Using Symbolic State Models", ACM Computer Architecture, Vol 45, No. 4, pp. 557-587, July 1998.