다중 유한상태머신 기반 병렬적 도달성 분석 기법

이정선, 이우진, 신영술, 카오 티 리
경북대학교 전자전기컴퓨터학부
e-mail : rubyindark@knu.ac.kr

A Parallel Reachability Analysis Method Based on Multiple Finite State Machine

Jung Sun Lee, Woo Jin Lee, Youngsul Shin, Cao Thi Ly
School of Electrical Engineering and Computer Science, Kyungpook National University

요 약

컴퓨팅 자원의 확보가 용이해짐에 따라 이러한 자원을 최대한 활용하려는 시도가 늘어나고 있다. 시스템을 검사하는 정형적 기법으로써 많이 사용되고 있는 모델 체킹은 상태폭발 문제를 완화하기 위해서 여러 컴퓨팅 자원을 한꺼번에 사용하려는 연구가 이루어져 왔다. 하지만 이 기법 역시 여러 상태 모델들이 하나로 합쳐지면서 여전히 상태폭발 문제를 발생 시킨다. 본 논문에서는 이러한 문제가 나타나는 원인을 지적하고 이를 해결하기 위해 모델 체킹의 기본 요소인 새로운 병렬적 도달성 분석 기법을 제시한다.

1. 서론

소프트웨어의 버그를 찾는 방법은 소스코드 작성 시점을 기준으로 소스코드가 작성된 후에 버그를 찾는 방법과 소스코드를 작성하기 전에 버그를 찾는 방법으로 나눌 수 있다. 전자의 경우에는 테스트와 시뮬레이션이 대표적이고 후자의 경우에는 정형적 검증 방법이 대표적이다. 정형적 검증 방법은 테스트 방법보다 이른 개발 단계에서부터 적용할 수 있어서 버그를 일찍 찾을 수 있고 그 비용이 상대적으로 적다는 장점 때문에 점점 더 많이 사용되고 있다.

모델 체킹(model checking)[1]은 대표적인 정형적 검증 방법 중 하나이다. 모델 체킹은 시스템의 상태를 나타낸 모델을 가지고 시스템이 원하는 속성을 만족하는지 검사하는 기술이다. 모델 체킹을 이용하기 위해서는 먼저 유한상태머신[2]이나 UML 상태머신 다이어그램[3] 등의 정형적 기법으로 시스템을 모델링해야 한다. 그 후에 이 모델을 가지고 시스템이 만족해야 하는 속성들을 검사하면서 문제점을 찾게 된다.

상태 공간을 순회하면서 간단한 검증이 가능한 도달성 분석은 모델 체킹 기술에서 아주 중요한 부분이다[4]. 시스템의 상태 공간은 여러가지 정형적 기법으로 나타낼 수 있는데, 검사할 때는 대부분 다루기 편리한 유한상태머신으로 변환하여 사용한다[5]. 이때 원래 사용했던 정형적 기법이 가지고 있던 구조적인 상태들이 평탄화되거나 시스템의 일부를 나타낸 여러 상태 모델들이 합쳐지면서 상태의 수가 기하급수적으로 늘어나는 상태 폭발 문제가 발생한다.

최근에는 컴퓨팅 자원의 확보가 용이해지면서 이러한 상태 폭발 문제를 완화하기 위해 모델 체킹을 병렬적으로 실행하는 방법도 연구되고 있다[4][6][7]. 하지만 여전히 상태폭발 문제는 모델 체킹의 적용에 큰 걸림돌이 되고 있다. 본 논문에서는 기존에 연구된 병렬적 도달성 검사가 가진 상태폭발 문제의 원인을 지적하고, 이 문제를 해결하기 위해서 새로운 병렬적 도달성 분석 기법을 제시한다.

논문의 구성은 다음과 같다. 먼저 2 장에서는 기존의 병렬적 도달성 분석에서 나타날 수 있는 상태 폭발 문제를 설명한다. 이어서 3 장에서 그러한 문제를 극복할 수 있는 새로운 기법을 설명한다. 4 장에서는 기존의 기법과 새롭게 제시되는 기법을 생각하는 철학자 문제를 이용해 비교해본다. 마지막으로 5 장에서는 결론을 맺고 연구 진행 방향에 대하여 논한다.

2. 병렬적 도달성 분석 기법에서의 상태 폭발

지금까지 연구된 병렬적 도달성 분석 기법은 전체 상태공간을 각각 하위의 부분 상태 공간으로 나누어 분석하는 방식을 기본으로 해왔다[7]. 그래서 부하를 골고루 나눌 수 있는 분할 함수를 연구하거나 프로세스간 통신 부하를 낮출 수 있는 분할 함수를 연구하는 것과 같이 상태 공간을 효율적으로 나누는 기법들이 연구되어 왔다[4][6]. 하지만 이 방식은 일반적인 도달성 분석 기법을 단순히 병렬적으로 변환시킨 것이다. 만약 여러 상태머신들의 조합으로 표현된 모델에 이 기법을 적용하려면, 우선 이 상태머신들을 하나의 상태머신으로 합성해야 한다. 이 과정에서 상태의 수가 기하급수적으로 늘어나는 상태 폭발 문제가 발생할 수 있다. 일반적으로 상태의 개수가 N 개인 상태머신 M 개를 하나의 상태머신으로 합성하게 되면,

※ 본 연구는 방위사업청과 국방과학연구소의 지원으로 수행되었습니다.

새롭게 만들어지는 상태머신이 가지는 상태의 개수는 N^M 개가 된다.

시스템이 커지고 복잡해지면, 하나의 큰 상태 머신으로 표현하기 보다는 각각의 하위 시스템으로 나누어진 상태 머신들로 표현하는 것이 자연스럽다. 그렇기 때문에 상태머신 하나만 가지고 분석하는 것은 문제가 있다. 따라서 이 문제를 해결하기 위해서는 기존의 기법과는 다르게 여러 상태머신들을 하나로 합성하지 않고 그대로 사용하는 기법이 필요하다. 본 논문에서는 이러한 상태 폭발 문제를 피할 수 있는 병렬적 도달성 분석에 사용할 새로운 상태머신 순회 기법을 제시한다.

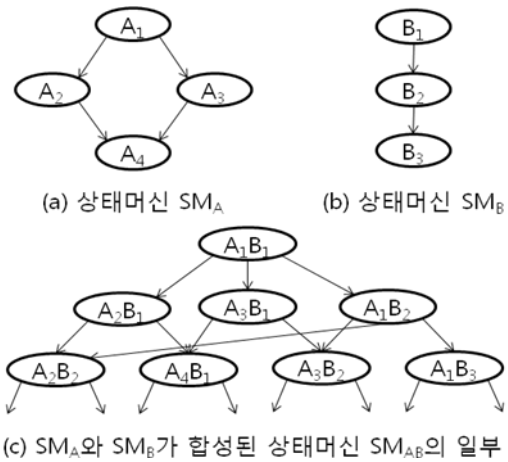
3. 병렬적 상태머신 순회 기법

본 논문에서 제시하는 병렬적 상태머신 순회 기법은 여러 상태머신들을 한번에 다루는 접근방법을 사용한다. 그러므로 이 상태머신들이 병렬적으로 실행될 때의 행동이 하나의 상태머신으로 합성되었을 때의 행동과 같도록 만들어야 한다. 그래서 다음과 같은 가정이 필요하다.

(1) 시스템의 현재상태는 모든 상태머신들의 현재상태의 조합으로 나타낸다.
(2) 시스템의 다음상태는 조합된 상태머신들의 현재상태 중 하나만 바뀐다.

아래의 그림 1 은 위의 가정을 토대로 두 상태 공간을 합성한 결과이다. 그림 1(a)와 (b)는 각각 상태머신 SM_A 와 SM_B 를 표현하고 있고, 그림 1(c)는 두 상태머신이 합성된 모습의 일부를 표현하고 있다. 이들이 하나의 상태머신 SM_{AB} 로 합성되면 현재 상태에서 다음 상태로 천이할 때 SM_A 혹은 SM_B 의 상태가 하나만 바뀐다.

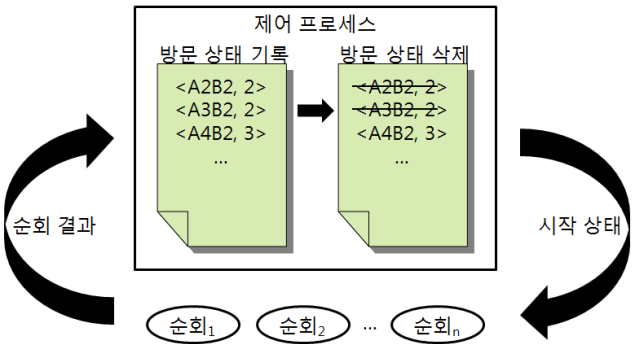
그림 1 의 SM_A 와 SM_B 를 각기 다른 순회 프로세스에서 순회하면서 SM_{AB} 를 순회하는 것과 같이 만들기 위해서는 이 프로세스들을 관리할 수 있는 제어 프로세스가 필요하다. 순회 프로세스는 제어 프로세스에게서 받은 상태를 시작으로 자신이 가진 상태머신만 이용해서 순회한다. 제어 프로세스는 순회 프로세스



(a) 상태머신 SM_A (b) 상태머신 SM_B

(c) SM_A와 SM_B가 합성된 상태머신 SM_{AB}의 일부

(그림 1) 상태 머신 합성의 예



(그림 2) 병렬적 상태 머신 순회 과정

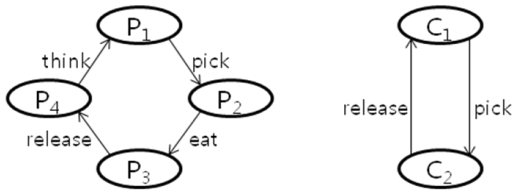
가 어떤 상태에서부터 순회를 시작해야 하는지를 관리하고, 순회 결과를 분석해서 다음 순회를 명령할 때 반영한다.

본 논문에서 제시할 기법의 기본 아이디어는 순회 과정에서 여러 번 방문할 수 있는 상태는 방문할 수 있는 횟수를 먼저 계산하고, 그 상태에 도달한 횟수가 계산된 횟수에 다다르면 기록을 삭제하는 것이다. 이렇게 함으로써 상태 폭발을 피하면서 모든 상태를 순회할 수 있다.

위의 그림 2 는 이 과정을 개략적으로 나타내고 있다. 먼저 제어 프로세스가 순회 프로세스에 순회를 시작할 상태로 초기 상태를 전송한다. 그러면 순회 프로세스는 받은 시작 상태에서부터 자신이 도달할 수 있는 상태를 모두 순회한다. 그리고 그 결과를 다시 제어 프로세스에 전송한다. 제어 프로세스는 이 결과를 분석한다. 순회 프로세스에서 방문한 상태가 처음 방문되었다면 그 상태와 그 상태에 도달할 수 있는 간선의 개수를 기록한다. 만약 방문한 상태가 간선의 개수만큼 방문되었다면 그 상태로는 다시 방문하지 않을 것이므로 기록에서 삭제한다. 분석이 완료되면, 제어 프로세스는 다음 순회 시작 상태를 결정하고 또다시 순회 프로세스에 시작 상태를 전송한다. 이 과정을 더 이상 순회를 시작할 상태가 없어질 때까지 계속하면 순회가 종료된다.

그림 1 의 경우를 예로 들면, 초기상태는 A_1B_1 이다. 이 상태가 각각 SM_A 와 SM_B 를 순회하는 프로세스로 전달된다. 그러면 SM_A 는 $\{A_2B_1, A_3B_1, A_4B_1\}$을 순회 결과로 제어 프로세스로 전송한다. 상태 A_2B_1 은 SM_A 의 상태 A_2 와 SM_B 의 상태 B_1 으로 구성된 상태이므로 이 상태 A_2B_1 에 도달할 수 있는 간선의 수는 A_2 에 도달할 수 있는 간선의 수와 B_1 에 도달할 수 있는 간선의 수의 합과 같다. 그러므로 상태 A_2B_1 에 도달할 수 있는 간선의 수는 1 이 되고, 이미 이 상태에 도달한 수 역시 1 이 되었으므로 이 상태는 기록되지 않는다. 이와 마찬가지 방법으로 A_3B_1 과 A_4B_1 역시 기록되지 않고, 이들은 나중에 SM_B 에서 순회의 시작 상태로 입력된다. SM_B 의 입력으로 받은 초기 상태 A_1B_1 으로 순회한 결과는 $\{A_1B_2, A_1B_3\}$가 되고, 이 역시 이전의 과정과 마찬가지로 기록되지 않고 나중에 SM_A 에서 순회의 시작 상태로 입력된다.

SM_A 는 새롭게 들어온 시작 상태 A_1B_2 에서부터 순회를 수행한다. SM_A 는 순회 결과로 $\{A_2B_2, A_3B_2, A_4B_2\}$



(a) 철학자의 상태머신 (b) 젓가락의 상태머신

(그림 3) 생각하는 철학자의 문제의 부분 상태머신

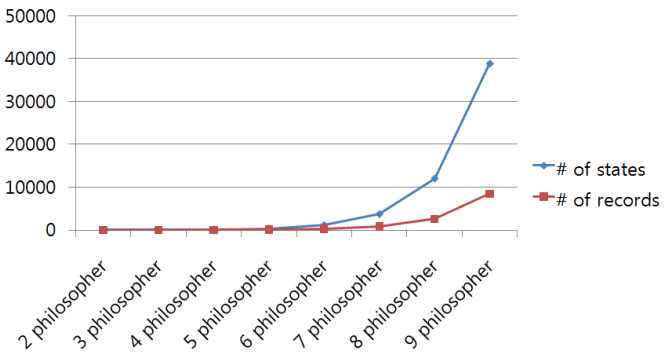
를 제어 프로세스로 전송한다. 상태 A_2B_2 에 도달할 수 있는 간선의 수는 2 개이므로 제어 프로세스에서 이 정보를 기록한다. 마찬가지로 상태 A_3B_2 와 A_4B_2 도 기록한다. SM_B 가 새롭게 들어온 시작 상태 A_2B_1 에서 순회를 시작하면 순회 결과로 $\{A_2B_2, A_2B_3\}$를 제어 프로세스로 전송한다. 상태 A_2B_2 는 도달할 수 있는 간선의 수가 2 개이지만 이미 SM_A 의 순회 결과로 등록되어있는 상태이다. 상태 A_2B_2 로 도달할 수 있는 간선의 수와 실제로 도달한 수가 같아졌으므로 제어 프로세스에서는 이 기록을 지우게 된다. 상태 A_2B_3 의 경우에도 같은 방식으로 처리된다.

4. 기존의 기법과의 비교

기존의 기법을 사용하기 위해 SM_A 와 SM_B 를 합성하여 SM_{AB} 를 만들면, 상태의 수는 12 개가 된다. 하지만 제시된 기법을 사용하면 SM_A 와 SM_B 를 합성하지 않게 된다. 게다가 SM_A 와 SM_B 를 순회하는데 걸리는 시간이 같다고 가정하면 제어 프로세스에서 기록하는 상태의 개수는 최대 3 개밖에 되지 않는다.

위의 그림 3 은 생각하는 철학자 문제를 단순화하여 나타낸 부분 상태머신들이다. 그림 3(a)는 철학자의 상태머신을 나타내고 있고, 그림 3(b)는 젓가락의 상태머신을 나타내고 있다. 여기에서 간선 pick 과 release 는 철학자와 젓가락이 함께 수행 되어야 한다. 예를 들어 철학자 A 가 상태 P_1 에서 P_2 로 천이되기 위해서는 양 옆의 젓가락들은 C_1 상태에 있어야 하고, 천이 되고 나서는 젓가락들의 상태도 C_2 로 천이 되어야 한다.

아래의 그림 4 는 생각하는 철학자 문제를 제시된 기법을 사용하여 순회한 결과로 나타나는 상태의 수와 순회 중 유지되는 최대 기록의 수를 그래프로 나타낸 것이다. 생각하는 철학자 문제에서 철학자의 수



(그림 4) 생각하는 철학자 문제의 상태 수 그래프

가 늘어나면 최종 합성된 상태머신의 상태 수는 기하급수적으로 늘어나는 것을 볼 수 있다. 하지만 제어 프로세스에서 기록하는 상태의 수는 총 상태의 수보다 완만하게 증가하는 것을 볼 수 있다. 그러므로 본 논문에서 제시된 기법이 기존의 방법보다 더 많은 상태를 다룰 수 있어 상태폭발을 완화시킨다고 볼 수 있다.

5. 결론

이 논문에서는 다중 유한상태머신을 대상으로 도달성 검사를 할 때, 기존의 병렬적 기법을 사용하면 유한상태머신을 하나로 합치는 과정에서 상태폭발 문제가 있을 수 있음을 지적하였다. 그리고 그 문제를 해결하기 위해 새로운 병렬적 도달성 검사 기법을 제시하였다. 제시된 기법은 다시 방문할 수 없는 상태를 검사하여 방문 후 이를 삭제함으로써 상태저장 공간을 줄일 수 있었다. 향후 연구에서는 이 기법을 토대로 병렬적 도달성 검사 도구를 구현하여 다른 기법과 비교할 수 있을 것으로 기대한다

참고문헌

[1] G.J. Holzmann, “The Model Checker SPIN,” *Transactions on IEEE Software Engineering*, Vol. 23, Issue 5, pp. 279 – 275, May 1997

[2] P.J. Denning, et al., *Machines, Languages, and Computation, Prentice Hall*, 1978

[3] http://www.omg.org/spec/UML/2.2/Superstructure

[4] D. Petcu, “Parallel Explicit State Reachability Analysis and State Space Construction,” *Proceedings of Second International Symposium on Parallel and Distributed Computing* (ISPDC 2003), pp. 207 – 214, 13-14 Oct. 2003

[5] 이우진, “Slice 모델을 이용한 유한상태머신의 트랜지션 축약 알고리즘,” 정보과학회논문지: 소프트웨어 및 응용, Vol. 5, No. 1, pp. 12 – 21, 2008 년 1 월

[6] M. Bourahla, M. Benmohamed, “Efficient Partition of State Space for Parallel Reachability Analysis,” *Proceedings of the 3rd ACS/IEEE International Conference on Computer Systems and Applications* (AICCSA 2005), Jan. 2005

[7] H. Garavel, R. Mateescu, I. Smarandache, “Parallel State Space Construction for Model Checking”, *Proceedings of 8th International SPIN Workshop*, LNCS #2057, pp. 217 – 234, Springer, 2001