2001년 한국정보처리학회 춘계 학술발표논문집 제8권 제1호

SPIN을 이용한 CTL 모델 체킹 방법론 연구

°방 기 석, 이 주 용, 최 진 영
고려대학교 컴퓨터학과

{kbang, jlee, choi}@formal.korea.ac.kr

CTL Model Checking using SPIN

°Ki-Seok Bang, Joo-Yong Lee, and Jin-Young Choi
Department of Computer Science and Engineering, Korea University

요 약

모델체킹에 사용되는 논리는 CTL 과 LTL 이 있다. 이 두 논리식은 그 표현력과 효과적인 면에서 크게 차이가 있다. 현재 SPIN 에서는 LTL 을 이용하여 모델체킹을 수행하고 있다. 본 논문에서는 LTL 모델체커인 SPIN 에 CTL 을 적용시킬 수 있는 가능성을 제시하고, LTL 모델 체커에 CTL을 적용시킬 수 있는 방법에 대해 논한다.

1. 서론

최근 들어 각종 시스템에 대한 안정성과 신뢰도를 높이기 위해 정형기법[1] 이 활발히 적용되고 있다. 특히, 모델 체킹 방법은 사용자가 이해하기 쉽고, 적용 분야가 매우 넓어 더욱 활발히 연구되고 있다.

모델 체킹[2] 방법에는 반드시 특정한 논리가 적용되어야 하는데, 현재 가장 활발히 사용되는 논리는 시제 논리(Temporal Logic)[3] ,[4] 이며, 그 중에서도 계산 트리 논리(CTL)[2] 과 선형 시제 논리(LTL)[4] 이 가장 많이 적용되고 있다. 그러나 두 논리는 공통으로 표현할 수 있는 특성보다는 자신만이 표현할 수 있는 영역이 훨씬 넓다. 따라서 CTL 모델 체킹[9] 과 LTL 모델 체킹[9] 방법 간에는 매우 많은 차이가 존재한다. 이러한 차이 때문에 모델 체킹을 사용하여 정형 검증을 실시할 때 도구 및 특성 표현의 제약이 주어질 수 있다. 따라서, LTL 과 CTL 을 제한하지 않고 모델 체킹에 적용할 수 있도록 하는 연구가 진행되고 있다. 본 연구에서는 LTL 모델체커인 SPIN 을 이용하여 CTL 모델체킹을 수행할 수 있는 방법에 대해 고안하고 그 가능성에 대해 논한다. 본 논문은 다음과 같이 구성된다. 2 절에서는 모델체킹과 논리식에 대해 설명하였고, 3 절에서는 LTL 모델체커인 SPIN 에 CTL 을 적용하기 위한 방법에 대해 논하며, 4 절에서 결론을 맺는다.

* 본 연구는 2000년 한국 과학 재단 특정기초 연구 지원사업(2000-1-30300-010-3)에 의해 연구되었습니다.

2. 관련 연구

(1) 정형기법 과 모델체킹

정형기법 (Formal Methods)[1] 은 컴퓨터 시스템을 개발하기 위해 수학적으로 분석하고 설계하는 기술을 의미한다. 이 정의에서 말하듯이 정형기법은 수학과 논리학에 기반을 둔 방법으로 하드웨어나 소프트웨어 시스템을 명세하고 시스템이 만족해야 할 특성 역시 수학적인 논리로 표현하여 시스템이 특성을 만족하는지를 증명한다. 이러한 정형기법은 수학적 모델을 설명하기 위한 문법, 그리고 문법의 의미를 설명하는 의미론 및 의미 관계를 포함한다. 따라서 정형기법은 자연어가 내포할 수 있는 애매모호함이나 불확실성을 최소한으로 줄일 수 있다. 또한 설계된 시스템이 처음에 의도된 요구사항과 동일한지를 수학적 성질을 이용하여 증명할 수 있기 때문에 개발초기에 큰 실수를 발견할 수 있다. 정형기법은 크게 정형 명세(Formal Specification)와 정형 검증(Formal Verification)의 두 가지로 구분할 수 있다. 정형명세는 시스템이 만족해야 할 요구사항과 그 요구사항을 만족할 수 있는 설계를 기술하는 것을 말한다. 정형검증은 명세가 정확한지, 즉 설계가 요구사항을 만족하는지를 검사하는 것이다

모델 체킹[2] 은 정형 검증 방법에서 사용하는 방법으로 시스템의 동작을 유한상태 기계의 형태로 명세하고 그 시스템이 만족해야 할 특성을 CTL 이나 LTL 과 같은 시제 논리로 표현한다. 그 후에 표현된 특성이 유한상태 기계로 명세 된 시스템에서 만족하는지를 검사하는 방법이다. 시제논리는 일반적으로 정리 증명에서 사용되는 논리에 비해 제한적이지만 간단하고 명확해서 유용하게 사용될 수 있다. 이 방법은 복잡한 시스템의 동작을 유한상태 기계로 표현함으로써 간단하게 표현할 수 있고, 이해하기가 매우 쉽다. 또한 모델 체킹용 도구가 제공되

어 명세 된 시스템에 대해 자동으로 검증을 수행할 수 있다. 그러나 이 방법 역시 시스템의 동작이 복잡하고 규모가 큰 경우에는 상태폭발 문제를 야기할 수 있다.

(2) CTL 과 LTL

모델 체킹에서 일반적으로 사용하는 논리는 명제논리(propositional logic)에 시간 연산자(temporal operator)를 넣어서 확장 시킨 분기시간 계산 트리 논리(Branching-time Computational Tree Logic : CTL)[4] 과 선형 시제 논리(Linear Temporal Logic : LTL)[4] 이다. CTL 은 시간의 흐름에 따라 발생할 수 있는 경로(path)에 대해서 트리 형태로 표현하는 것이다. 계산 트리는 상태 전이 그래프에서 유도 된다. 이 트리에서 나타나는 모든 경로는 모델링 되는 시스템의 모든 가능한 계산을 표현하고 있다. CTL 은 이러한 트리와 같은 분기 구조를 묘사하는 연산자(operator)를 가지고 있기 때문에 분기시간논리(branching time logic)로 분류된다. CTL 로 표현되는 식은 단위 명제(atomic proposition)와 명제논리의 부울 합성자(boolean connectives), 시제 연산자(temporal operator)로 구성된다

먼저 proposition 은 상태 변수(state variable)들의 불린 조합(boolean combination)이다. 그리고, formula 는 proposition 이거나, formula 의 불린 조합이다. 또한, f 가 formula 라면, AG f, AF f, EG f, EF f 의 불린 조합도 formula 이다. 각 formula 가 q 라는 상태(state)에서 적용된다고 가정하면, 연산자 A 는 q 에서 시작하는 모든 경로라는 의미이고, E 는 q 에서 시작하는 어떤 한 경로라는 뜻이 된다. 또한, G 는 해당 경로의 모든 상태를 의미하는 것이며, F 는 해당 경로의 어떤 한 상태를 의미하는 것이다. 그래서, AG safe 가 q 에서 만족되려면, q 에서 시작하는 모든 경로(A)의 모든 상태(G)들이 safe 라는 formula 를 만족해야 한다. 또한, AF safe 가 q 에서 만족되려면, q 에서 시작하는 모든 경로(A)에 대해, 각 경로에서 적어도 하나의 상태(F)가 safe 를 만족해야 하며, EG safe 가 q 에서 만족되려면, q 에서 시작하는 경로 중 적어도 하나의 경로(E)의 모든 상태(G)가 safe 를 만족해야 한다. 마지막으로, EF safe 가 q 에서 만족되려면, q 에서 시작하는 경로 중 적어도 하나의 경로(E)에서 적어도 하나의 상태(F)가 safe 를 만족해야 한다.

LTL 은 어떤 시스템의 동작을 상태의 시간적 흐름에 따라 선형적으로 표현을 하는 것이다. LTL 에서의 시간의 구조는 전순서집합(totally ordered set) (S, <)이다. 또한 자연수의 집합과 아이소모픽(isomorphic)하다. 또한 시간은 이산적이며, 초기상태가 있으며 무한히 계속된다. 선형시간 구조(linear-time structure) M=(S, x, L)로 정의된다. 여기에서 S 는 상태의 집합이며, x 는 자연수에서 S 로 대응되는 무한한 상태의 흐름이다. 그리고 L 은 S 에서 단일명제기호(atomic propositional symbol)의 부분집합으로 대응되는 함수이다. 명제시제논리의 기본적인 연산자는 다음과 같다.

Fp : 언젠가 p가 참이다.
Gp : 언제나 p는 참이다.
Xp : 다음 시점에 p가 참이 된다.
p U q : 현재 q가 참이거나 q가 참일 시점까지 p가 참이다.

예를 들어 $p \Rightarrow Fp$의 시제논리식은 "만일 p가 현재 시점에서 참이라면 미래의 어떤 시점에 q가 참이 될 것이다."라는 의미를 갖게 된다.

3. SPIN 과 CTL 의 적용 방법

(1) SPIN 에서의 LTL 모델체킹

SPIN[5] 은 AT&T Bell 연구소에서 1995 년에 발표한 LTL 모델체커이다. 기존의 모델 체킹 방법을 개선하기 위해 On-the-Fly 방식을 사용하여 메모리의 효율적인 사용을 가능하도록 하였다. SPIN 은 비동기적 프로세스 시스템(asynchronous process system)의 설계와 검증(verify)을 지원하는 가장 일반적인 tool 이다. SPIN 은 또한 분산 소프트웨어(distributed software)와 통신 프로토콜(communication protocol) 검증에 아주 유용하게 사용되고 있다. SPIN 검증 모델은 프로세스 상호작용(Process interactions)의 정확성(correctness)에 초점을 둔다 SPIN 은 PROMELA(Process Meta LAnguage)라는 검증 언어를 사용하여 설계를 하고 PROMELA[6] 는 LTL 의 구문으로 명세화 하여 정확성을 증명한다. PROMELA 는 검증 모델 언어(Verification modeling language)이고, 분산 시스템(distributed system)이나 프로토콜의 추상모델(abstraction)을 제공하고 프로세스(processes), 메시지 채널(message channels), 그리고 변수들로 구성되어 있다.

SPIN 은 LTL 로 표현된 특성을 Buchi 오토마타의 형태로 변환시킨다. 이 오토마타는 유한한 상태를 가지며 무한한 입력에 대한 동작을 표현한다. 따라서, 어떤 특성이 만족상태(acceptance state)를 무한히 자주 방문할 때 그 특성이 모델을 만족한다고 말할 수 있다. 이렇게 변환된 Buchi 오토마타를 다시 Promela 의 형태로 변환하여 검증을 실행하게 된다. 간단한 변환 예는 다음과 같다.

```
never {    /* []p -> <>q */
T0_init:
    if
    :: ((! ((p)) || (q))) -> goto accept_all
    :: (1) -> goto T0_init
    fi;
accept_all:
    skip
}
```

그림 1 Promela 로 변환된 LTL

SPIN 은 LTL 로 명세된 특성이 입력되면 이것의 역을 구해 Buchi 오토마타의 형태로 변환 시킨다[8] . 그리고 이것을 다시 Promela 로 생성시켜 검증을 한다. 즉, 모델 역시 Buchi 오토마타의 형태이고 특성 역시 Buchi 오토마타의 형식이다.

그 후에 모델의 오토마타와 특성의 역에 대한 오토마타의 교집합을 구한다. 이 때 교집합이 공집합이면 그

모델은 특성을 만족한다고 할 수 있으며, 어떤 형태의 교집합이 생성되게 되면 그것은 반례로 출력되게 된다 [10] .

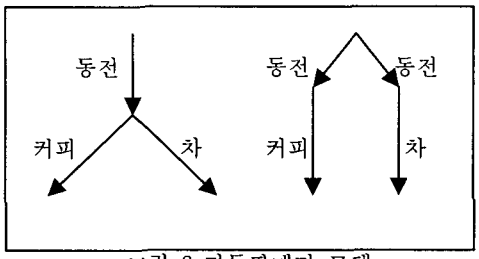

그림 2 자동판매기 모델

(2) CTL 의 적용 방법

그림은 간단한 커피 자동판매기의 모델[10] 이다.

그림에서 보는 바와 같이 두 자동판매기는 동전을 넣으면 커피 혹은 차를 먹을 수가 있다. 두 모델에서의 연산 상태 집합은 {(coin, coffee), (coin, tea)}로 동일하다. 그러나, 두 모델의 동작은 확실히 차이가 난다. 왼쪽의 모델의 경우 동전을 넣은 후 커피나 차를 선택할 수 있다. 그러나 오른쪽의 모델에서는 동전을 넣으면 비결정적으로 커피나 차가 배출되게 된다. 따라서 사용자의 요구에 상관없이 비결정적으로 상태가 전이되게 된다. 위의 모델을 SPIN 으로 모델링 하면 다음과 같이 비교된다.

```
                ...
:: (coin == 1) && (coffee == 1)->
atomic
{
    coffee_out = 1;
    coin = 0; coffee = 0;
    coffee_out = 0;
}
                ...
```

```
                ...
:: (coin == 1) ->
atomic
{
    coffee_out = 1;
    coin = 0;
    coffee_out = 0;
}
                ...
```

그림 3 Promela 로 모델링 된 자동판매기

위에서 볼 수 있듯이 표의 윗부분이 제대로 동작하는 자동판매기의 커피 판매 부분이고 아래부분이 비결정적인 동작을 하는 자동판매기의 부분이다. 두 모델이 서로 다르게 명세는 되었지만 실제로 LTL 을 이용하여 특성을 모델링 할 경우에는 두 모델을 구분할 수가 없다. 즉, 상태에 대한 정보가 없기 때문에 (동전 -> 커피) or (동전 -> 차) 인 동작만을 구분할 수 있다. 그러나, CTL 을 이용하여 특성을 명세할 경우에는 두 모델에서 확실한 차이가 있다. "동전을 넣으면 다음에는 반드시 커피를 먹을 수가 있다."라는 동작을 CTL 로 표현하면 "G(coin -> EX coffee)로 명세할 수 있다. 이런 특성을 두 자동판매기 모델에 적용시켜 보면 오른쪽의 자동판매기는 만족시킬 수 없다.

위에서 볼 수 있듯이, 분기가 발생했을 때의 조건을 판단할 경우에는 CTL 이 훨씬 그 효율이 좋다고 볼 수 있다[7] .

본 연구에서는 위와 같은 경우에 대해 적용할 수 있도록 CTL 을 LTL 모델 체커인 SPIN 에 적용하기 위해 몇 가지 가능한 방법을 제안해 보도록 한다.

첫번째로 CTL 을 LTL 로 변환시켜 SPIN 에 적용하는 방법이다. 이 방법은 Cadence SMV[12] 에서 LTL 을 적용하는 방법에서 착안한 것이다[11] . Cadence SMV 는 기존의 SMV 가 CTL 을 입력 받아 검증하는 방식을 사용하는 대신 LTL 을 입력 받을 수 있도록 고안했다. 그러나 이 방법 역시 제약이 있으며 현재도 계속 연구가 진행중이다. 그 이유는 다음과 같다.

LTL 과 CTL 은 그 검증 대상과 표현 방법에 있어 매우 큰 차이가 있다. 따라서 LTL 모델체커인 SPIN 에 CTL 을 적용하기는 매우 어렵다. 우선 LTL 과 CTL 은 표현할 수 있는 특성이 서로 다르다. 다음의 다이어그램을 보자.

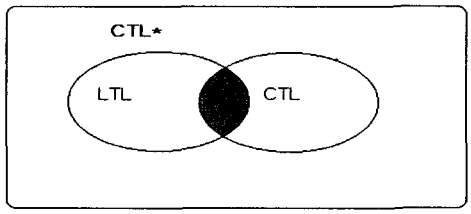

그림 4, CTL*, LTL, CTL 의 표현력

위의 다이어그램을 보면 알 수 있듯이 LTL 과 CTL 이 모두 만족시킬 수 있는 특성이 있는 반면에 서로가 만족시키지 못하는 특성이 존재하게 된다. 즉, CTL 로만 표현 가능한 특성이 반드시 존재하게 된다. 예를 들면 CTL 의 AGp 와 같은 특성은 "모든 경로에서 언제나 p 를 만족한다."를 뜻한다. 이 특성은 LTL 에서 Gp 로 표현을 하더라도 LTL 의 문법적인 정의에 어긋나지 않는다. 실제로 LTL 은 A 라는 경로 한정자를 내포하고 있다고 정의한다. 그러나 CTL 의 E 라는 경로 한정자를 정의하지 않고 있기 때문에 "어떤 경로에서 언젠가 p 를 만족한다."를 나타내는 EFp 와 같은 특성은 LTL 로 표현을 할 수 없다. 따라서 CTL 을 LTL 로 변환시켜 SPIN 에 적용하는 방법은 많은 제약점과 문법적인 문제점이 존재할 수 있다.

두 번째로 생각할 수 있는 것이 CTL 을 Buchi 오토마타의 형태로 변환시키는 방법이다. 이 방법은 SPIN 에서 사용하는 방법을 적용시키는 것이다. 앞에서 보였듯이 SPIN 에서는 기본적인 LTL 모델체킹 방법을 따르고 있다. 즉, LTL 을 표현한 오토마타와 모델 오토마타를 이용하여 원하는 특성을 만족하는지 증명한다. CTL 을 입력 받았을 때 이 방법을 적용하는 것이다. 그러나 실제로 모든 CTL 을 Buchi 오토마타의 형태로 변환시키려는 것은 아니다. 일반적으로 모델체킹을 수행할 때 관심 있는 특성의 대부분은 공정성이다. 예를 들어 통신 프로토콜을 검증하고자 할 때 "계속해서 전송되지만 절대로 수신되지 않는 메시지는 없다."라는 특성은 대표적인 공정성 특성을 보여준다. 여기서 말하는 공정성 특성이란 만일 공정성 조건이 상태의 집합으로 표현된다면, 공정한 경로는 반드시 각 공정성 조건을 무한히 자주 반복해서 만족시켜야 한다. 이런 공정성 조건이 CTL 로 표현되어 있다면 그 경로는 각 조건이 통해 무한히 자주 경로상에 나타나면 그 경로는 공정하다고 할 수 있다. 이 때 CTL 식의 경로 한정자는 역시 공정한 Kripke 구조를 갖는다고 할 수 있다. CTL 은 Kripke 구조를 표현할 수 있으며, M=(S,R,L,F) 의 네 개의 쌍으로 이뤄진다. 이 때 $F \subseteq 2^S$ 는 공정성 조건의 만족 조건이다. 이 때 이런 조건을 일반화된 Buchi 만족 조건이라고 한다. 일단 CTL 이 일반화된 Buchi 오토마타의 형태로 변환이 가능하면 그 후에는 SPIN 에서 사용하는 방법을 이용하여 해당 오토마타에 맞는 Promela 코드로 생성할 수 있다. 즉, 주어진 CTL 에 별다른 변환 과정 없이 바로 Promela 코드로의 변환이 가능하기 때문에 LTL 로의 변환보다 제약이 적고, 적용 가능한 경우도 많아질 것으로 예상하고 있다.

4. 결론 및 향후 계획

현재 많은 분야에서 정형기법을 이용하여 시스템의 신뢰도와 안정성을 높이려는 노력을 하고 있다. 정형기법 중의 한 분야인 모델 체킹 방법을 이용하면 사용자가 이해하기 쉽고, 자동화된 도구를 이용하여 빠르게 정형검증을 실시할 수 있다. 그러나, 모델 체킹을 적용할 때 CTL 혹은 LTL 을 사용하여 시스템의 특성을 표현하고자 할 때 그 선택에 대한 문제가 시스템에 따라 매우 민감하게 대두될 수 있다. 두 논리식이 서로 다른 표현 능력과 적용 분야를 보이고 있기 때문에 적절한 논리식의 선택이 검증의 범위와 결과에 큰 영향을 줄 수 있기 때문이다. 본 연구에서는 이런 영향을 줄이고 논리 선택을 자유롭게 하기 위한 목표로 LTL 모델 체커인 SPIN 에서 CTL 을 사용할 수 있는 방법론을 구상하였다. 현재, LTL 의 일부를 CTL 로 표현하여 SMV 에 적용하려는 연구 역시 활발히 진행하고 있다. SPIN 은 구현이 매우 쉽고 시뮬레이션 및 검증을 모두 제공하기 때문에 그 사용 범위가 매우 다양하다. 따라서 몇 가지 제약이 있지만 CTL 을 SPIN 모델 체커에 적용할 수 있다면 그 검증 능력을 한층 높일 수 있으리라 생각한다. 현재는 CTL 의 적용가능성을 타진하고 CTL 의 LTL 변환 및 Buchi 오토마타로의 변환에 대해 연구 중이며, 향후 SPIN 에서 사용하는 경로 탐색 알고리즘을 개선하여 CTL 을 직접 적용할 수 있는 방법에 대한 연구도 병행할 예정이다.

참고 문헌

[1] Edmund M. Clarke and Jeannette M. Wing, "Formal Method : State of the Art and Future Directions", ACM Computing Surveys, pp. 626-643

[2] K. L. McMillan, *SYMBOLIC MODEL CHECKING*, Kluwer Academic Publisher, 1993.

[3] A. Pnueli, "The Temporal Logic of Programs," Proceedings of the 18th IEEE symposium Foundations of Computer Science, 1977.

[4] Zohar Manna, Amir Pnueli, "The Temporal Logic of Reactive and Concurrent Systems - Specification", Springer-Verlag, 1996

[5] Gerard J. Holzmann, "The Model Checker SPIN," IEEE Transactions on Software Engineering, May 1997.

[6] Gerard J. Holzmann, Design and Validation of Computer Protocols, Prentice Hall, 1991.

[7] O. Kupferman, M.Y. Vardi,"Relating Linear and Branching Model Checking", proceedings of the PROCOMET'98, 1998

[8] R. Gerth, D. Peled, M.Y. Vardi, P. Wolper, "Simple On-the-fly Automatic Verification of Linear Temporal Logic", proceedings of the PSTV'95, 1995

[9] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled, *Model Checking*, The MIT Press, 1999.

[10] Markus Muller-Olm, David Schmidt, and Bernhard Stefen, "Model-Checking : A Tutorial Introduction", Springer LNCS 1694, 1999, pp. 330-354.

[11] Edumund M. Clarke, O. Grumberg, K. Hamaguchi, "Another Look at LTL Model Checking", Formal Methods in System Design, 10:47--71, 1997.

[12] http://www-cad.eecs.berkeley.edu/~kenmcmil/smv/