모델 체킹에서 상태 투영을 이용한 모델의 추상화

권 기 현†

요 약

지금까지 제시된 정형 검증 기법들 중에서 모델 체킹이 가장 효과적이라는 평가를 받고 있지만, 모델 체킹이 실제 활용되기 위해서는 상태 폭발 문제를 극복해야 한다. 본 논문에서는 상태 폭발 문제를 방지하기 위해서, 원래 모델 M으로 부터 추상화 모델 M'을 얻는 방법을 제안하였다. 주어진 논리식의 모델 체킹에 필요한 변수만을 추출한 후, 모델의 상태 공간을 이들 변수들에 투영함으로서 추상화 모델 M'을 얻었다. M'은 M보다 크기가 작을뿐만 아니라 더 적은 행위를 갖고 있기 때문에($M' \preceq M$), 추상화 모델 M'을 이용해서 수행한 도달성 분석 결과는 M에서도 그대로 유효하다. 따라서 M을 모델 체킹할 때 상태 폭발이 발생하면, 축소된 모델 M'을 이용하여 모델 체킹할 수 있다. 제안된 추상화 기법을 푸쉬 푸쉬 게임 풀이에 적용했고, 모델 체커로는 Cadence SMV와 NuSMV를 사용하였다. 그 결과 상태 폭발 문제로 인해서 풀 수 없었던 게임을 추상화를 이용해서 해결하였다. 그리고 추상화를 적용하기 이전에 비해서 시간 절감 및 메모리 절감 효과가 있었다. Cadence SMV의 경우 평균 87%의 시간 절감 및 79%의 메모리 절감이 있었으며, NuSMV의 경우 83%의 시간 절감 및 56%의 메모리 절감이 있었다.

Abstraction of Models with State Projections in Model Checking

Gihwon Kwon†

ABSTRACT

Although model checking has gained its popularity as one of the most effective approaches to the formal verification, it has to deal with the state explosion problem to be widely used in industry. In order to mitigate the problem, this paper proposes an abstraction technique to obtain a reduced model M' from a given original model M. Our technique identifies the set of necessary variables for model checking and projects the state space onto them. The abstract model M' is smaller in both size and behavior than the original model M, written $M' \preceq M$. Since the result of reachability analysis with M' is preserved in M, we can do reachability analysis with model checking using M' instead of M. The abstraction technique is applied to Push Push games, and two model checkers - Cadence SMV and NuSMV - are used to solve the games. As a result, most of unsolved games with the usual model checking are solved with the abstraction technique. In addition, abstraction shows that there is much of time and space improvement. With Cadence SMV, there is 87% time improvement and 79% space one. And there is 83% time improvement and 56% space one with NuSMV.

키워드 : 키워드 : 모델 체킹(Model Checking), 추상화(Asbtraction), 축소(Reduction), 상태 폭발 문제(State Explosion Problem)

1. 서 론

CTL(Computation Tree Logic) 모델 체킹은 유한 상태 모델 M과 CTL 논리식 ϕ을 받아서, 이들간의 만족성 관계 $M \models \phi$를 결정한다[1]. 모델이 갖는 모든 상태 공간을 철저하게 탐색하면서 모델과 논리식 간의 만족성 관계를 결정하기 때문에, 기존의 테스팅이나 시뮬레이션으로 찾을 수 없는 에러들을 모델 체킹으로 찾을 수 있다. 그리고 검사 과정이 사용자의 개입없이 자동 처리되기 때문에, 정리 증명과 같은 다른 정형 검증 기법에 비해서 빠르고 사용하기 쉽다. 한편, 모델이 논리식을 만족하지 않는 경우, 모델 체킹은 그 이유를 담은 반례를 제공함으로서 모델의 디버깅 작업을 돕는다. 이런 이점들 덕분에 지금까지 제시된 정형 검증 기법들 중에서 모델 체킹이 가장 효과적이라는 평가를 받고 있다[2].

한편, 모델 체킹이 극복해야할 장애물도 여전히 많이 존재한다. 모델 체킹의 대표적인 문제점은 상태 폭발 문제이다. 전술한 바와 같이 모델 체킹의 기본은 상태 공간 탐색에 있다. 모델이 갖는 상태 공간을 철저하게 탐색하기 때문에, 모델의 크기가 커질수록 검사해야 할 상태 공간의 수는 지수적으로 증가한다. 이것을 상태 폭발 문제라고 부르며, 크기가 큰 모델을 성공적으로 모델 체킹하기 위해서는 상태 폭발 문제를 해결해야 한다[3]. 상태 폭발 문제를 방지하기 위해서 본 논문에서는 추상화를 사용한다. 원래 모델 M이 주어졌을때, 추상화의 목표는 M보다 크기가 더 적은 추상화된 모델 M'을 찾는 것이며, 이때 추상화된 모델 M'의 모델 체킹 결과는 원래 모델에서도 보존되어야 한다[4]. 다시 말해서, 추상화된 모델 M'은 M보다 크기가 적을뿐만 아니라 M'의 모델 체킹 결과를 이용해서 원래 모델 M의 모델 체킹 결과를 예측할 수 있어야 한다.

※ 본 연구는 2004학년도 경기대학교 학술연구비(일반연구과제) 지원에 의하여 수행되었음.
† 종신회원 : 경기대학교 정보과학부 교수
논문접수 : 2004년 7월 12일, 심사완료 : 2004년 8월 25일

본 논문에서는 상태 공간의 투영을 통해서 추상화된 모델 M'을 얻는 방법을 제안한다. 제안한 방법으로 얻어진 추상화된 모델 M'은 원래 모델 M보다 크기가 작을뿐만 아니라 더 적은 행위를 갖는다 ($M' \lesssim M$). 적은 행위를 갖는 모델에서 ECTL[1] 논리식이 만족되면, 더 큰 행위를 갖는 모델에서도 ECTL 논리식이 만족된다는 것이 판명되었다[5]. 상태 폭발 문제가 발생해서 주어진 모델로는 ECTL 논리식의 만족성 여부를 결정할 수 없을때, 추상화된 모델 M'를 이용해서 대신 모델 체킹할 수 있다. 추상화된 모델은 원래 모델에 비해서 크기가 축소되었기 때문에 상태 폭발 문제가 발생될 가능성도 그 만큼 줄어든다.

본 논문에서 제안한 추상화 기법을 푸쉬 푸쉬 게임 풀이에 적용하였다. 푸쉬 푸쉬 게임은 도달성 문제로 간주할 수 있기 때문에, ECTL 모델 체킹으로 게임을 풀 수 있는 최단 경로를 얻을 수 있다[6]. 푸쉬 푸쉬 게임 풀이에 추상화를 적용했을때 상태 폭발 문제가 얼만큼 개선되었는지를 실험하기 위해서 두 개의 모델 체커를 사용하였다. Cadence SMV[7]를 사용한 경우를 설명하면 다음과 같다. 추상화없이 전체 50게임 중에서 37게임을 풀었으나, 나머지 13게임은 상태 폭발 때문에 풀지 못했다. 추상화를 적용한 경우 추상화 이전에 비해서 13%의 시간과 21%의 메모만을 사용하고서도 37게임에 대해서 똑 같은 결과를 얻었다. 즉 87%의 시간 절감 및 79%의 메모리 절감 효과가 있었다. 뿐만 아니라, 상태 폭발 문제가 발생된 13게임 중에서 12게임을 추상화를 적용해서 해결하였다. NuSMV[8]의 결과도 비슷하다. 추상화없이 전체 50게임 중에서 43게임을 풀었으나, 나머지 7게임은 상태 폭발 때문에 풀지 못했다. 추상화를 적용한 경우 추상화 이전에 비해서 17%의 시간과 44%의 메모만을 사용하고서도 43게임에 대해서 똑 같은 결과를 얻었다. 즉 83%의 시간 절감 및 56%의 메모리 절감 효과가 있었다. 뿐만 아니라, 상태 폭발 문제 때문에 해결하지 못했던 7게임중에서 6게임을 추상화를 적용해서 해결하였다. 실험 결과에서 보듯이 추상화는 시간과 메모리 절감을 제공할 뿐만아니라 크기가 큰 모델의 모델 체킹을 가능케 했다.

논문의 구성은 다음과 같다. 2장에서는 모델 체킹 및 모델간의 관계에 대해서 살펴본다. 3장에서는 본 논문에서 제안하는 추상화 기법을 설명한다. 그리고 4장에서는 추상화를 적용해서 상태 폭발 문제를 해결한 사례를 밝히고, 추상화 이전에 비해서 시간과 메모리가 얼만큼 절감되었는지를 실험 결과를 통해서 설명한다. 5장에서 추상화와 관련된 연구를 살펴보고, 결론 및 향후 연구 과제를 마지막 6장에서 기술한다.

2. 배경지식

본 장에서는 모델 체킹과 관련된 배경 지식을 간략히 기술한 후, 모델간의 두 가지 관계인 바이시뮬레이션과 시뮬레이션 관계에 대해서 살펴본다. 지면의 제약상 본 논문의 전개에 필요한 부분만 언급할 것이며, 보다 상세한 사항은 [9]를 참고하기 바란다.

2.1 모델 체킹

CTL 모델 체킹은 유한 상태 모델 M과 CTL 논리식 ϕ을 입력받아서, 둘간의 만족성 관계 $M \vDash \phi$를 결정한다. 유한 상태 모델 $M = (S, I, R, AP, L)$은 5-튜플로 구성된다. 여기서 S는 상태들의 집합, $I \subseteq S$는 초기 상태들의 집합, $R \subseteq S \times S$는 상태들 간의 전이 관계, AP는 단순 명제들의 집합, $L : S \rightarrow 2^{AP}$는 각 상태에서 참이 되는 단순 명제들을 해당 상태에 배정하는 함수이다. 모델 체킹은 시스템이 갖는 무한 행위에 대해서 조사를 하기 때문에 상태들 간의 전이를 나타내는 R은 전체 관계이다. 즉 $\forall s \in S \bullet \exists s' \in S \bullet (s, s') \in R$로서 모든 상태마다 전이할 수 있는 다음 상태가 최소한 하나 이상 존재한다. 경로 $\pi = s_1 s_2 s_3 s_4 \cdots$는 전이 가능한 상태들을 차례대로 나열한 것으로서 $(s_i, s_{i+1}) \in R$, $i \geq 1$이며 그 길이는 무한이다.

모델에 관한 속성은 모델을 트리의 관점에서 해석하는 CTL 논리식으로 표현한다. 모델의 초기 상태를 루트로 해서 모델을 풀어헤치면 무한 트리를 얻게 되며, 트리의 각 경로는 모델의 행위를 나타낸다. 그러므로 트리는 모델의 가능한 행위를 모두 갖는다. 모델의 속성을 정형적으로 기술하기 위해서 CTL은 두 개의 경로 한정자 A(All), E(Exist)와 네 개의 시제 연산자 X(neXt), F(Future), G(Globally), U(Until)를 갖는다. 경로 한정자와 시제 연산자를 조합하면 8개의 연산자 AX, EX, AF, EF, AG, EG, AU, EU를 가지며, 이들을 사용하여 다양한 논리식을 기술한다. 예를 들어 단순 명제 $p \in AP$가 주어졌을때, CTL 논리식 AG p 의미는 '도달 가능한 모든 상태에서 p가 참이다'이며, EF p의 의미는 '언젠가는 p가 참인 상태로 도달 가능하다'이다. 모델 M의 상태 s에서 CTL 논리식 ϕ가 참인 경우를 $M, s \vDash \phi$로 표시한다. 상태 s에서 AG p와 EF p의 정형적 의미는 다음과 같다.

$$M, s \vDash \text{AG } p \quad \text{iff } \forall \pi = s_1, s_2, \cdots \forall i \geq 1 \cdot M, s_i \vDash p$$
$$M, s \vDash \text{EF } p \quad \text{iff } \exists \pi = s_1, s_2, \cdots \exists i \geq 1 \cdot M, s_i \vDash p$$

여기서 $s = s_1$이다. 모델 M의 모든 초기 상태에서 ϕ가 참인 경우를 $M \vDash \phi$로 표시하며 'M이 ϕ를 만족한다'라고 읽는다. 그렇지 않는 경우 $M \nvDash \phi$로 표기한다.

2.2 모델간의 관계

상태 폭발 문제를 해결하는데 가장 기본적으로 사용되는 것이 추상화이다. 전술한 바와 같이, 추상화의 목표는 원래 모델 M보다 크기가 더 적으면서도 원래 모델의 행위를 보존하는 M'을 찾는 것이다. 행위 보존을 고려할 때 원래 모델 M과 축소된 모델 M' 사이에는 두 가지 관계가 있다. 하나는 행위가 강하게 보존되는 바이시뮬레이션 관계이며($M \equiv M'$), 다른 하나는 행위가 약하게 보존되는 시뮬레이션 관계이다($M \lesssim M'$).

행위가 두 모델 사이에서 강하게 보존되는 관계를 먼저

1) CTL 논리식의 일부분으로서 "어떤 경로가 존재한다"를 나타내는 E가 시제 연산자 앞에 나오는 논리식이다.

살펴보자. 두 모델 $M=<S, I, R, AP, L>$과 $M'=<S', I', R', AP', L'>$이 있다고 하고, $AP=AP'$로서 두 모델의 기본 명제 집합이 같다고 하자. 모든 상태 s, s'에 대해서, 만약 $B(s,s')$라면 다음 다섯 가지 조건을 만족할 때 관계 $B \subseteq S \times S'$는 두 모델 M과 M'의 바이시뮬레이션 관계라고 한다.

① $L(s) = L'(s')$.
② $R(s,t)$인 모든 상태 t에 대해서 $R'(s',t') \wedge B(t,t')$을 만족하는 상태 t'가 존재한다.
③ $R'(s',t')$인 모든 상태 t'에 대해서 $R(s,t) \wedge B(t,t')$을 만족하는 상태 t가 존재한다.
④ M의 모든 초기 상태 $s_0 \in I$에 대해서 $B(s_0, s'_0)$을 만족하는 초기 상태 $s'_0 \in I'$가 M'에 존재한다.
⑤ M'의 모든 초기 상태 $s'_0 \in I'$에 대해서 $B(s_0, s'_0)$을 만족하는 초기 상태 $s_0 \in I$가 M에 존재한다.

조건 ①은 배정된 단순 명제들이 같아야 함을 나타내며 조건 ②, ③은 현재 상태에서 다음 상태로의 움직임을 다른 모델이 그대로 흉내낼 수 있는지를 보인다. 그리고 조건 ④, ⑤는 다른 모델이 초기 상태를 흉내낼 수 있는지를 보인다. 위의 다섯 가지 조건을 모두 만족할 때 모델의 행위는 강하게 보존된다. 왜냐하면 M의 행위를 M'이 흉내낼 수 있고, 거꾸로 M'의 행위를 M이 흉내낼 수 있기 때문이다. 따라서 두 모델은 바이시뮬레이션 관계에 있으며 $M \equiv M'$이라 표기한다.

바이시뮬레이션은 이상적이기는 하지만 상태 축소의 효과를 크게 기대할 수 없다. 보다 큰 상태 축소 효과를 얻기 위해서는 한 방향의 희생이 필요하다. 이를 위해서 행위가 약하게 보존되는 관계를 살펴보자. 두 모델 $M=<S, I, R, AP, L>$ 과 $M'=<S', I', R', AP', L'>$이 있고, 기본 명제 집합이 $AP \supseteq AP'$라고 하자. 모든 상태 s, s'에 대해서, 만약 $B(s,s')$라면 다음 세 가지 조건을 만족할 때 관계 $B \subseteq S \times S'$는 두 모델 M과 M'의 시뮬레이션 관계라고 한다.

① $L(s) \cap AP' = L'(s')$.
② $R(s,t)$인 모든 상태 t에 대해서 $R'(s',t') \wedge B(t,t')$을 만족하는 상태 t'이 존재한다.
③ M의 모든 초기 상태 $s_0 \in I$에 대해서 $B(s_0, s'_0)$을 만족하는 초기 상태 $s'_0 \in I'$가 M'에 존재한다.

위의 세 가지 조건을 모두 만족할 때 모델의 행위는 약하게 보존된다. 왜냐하면 M의 행위를 M'이 흉내낼 수 있지만, 거꾸로 M'의 행위를 M이 흉내내지는 못한다. 따라서 두 모델은 시뮬레이션 관계에 있으며 $M \lesssim M'$이라 표기한다. 시뮬레이션은 바이시뮬레이션에 비해서 한 방향을 양보했기 때문에 상태 축소 효과가 높다. 이러한 이유로 상태 폭발을 방지하는데 있어서 바이시뮬레이션 보다는 시뮬레이션이 많이 활용되고 있다.

$M \equiv M'$인 경우 CTL 논리식 ϕ에 대해서 $M \models \phi \Leftrightarrow M' \models \phi$이 성립한다. 즉 M에서 ϕ가 만족되면 M'에서도 ϕ가 만족되고, 거꾸로 M'에서 만족되면 M에서도 만족된다. 왜냐하면 두 모델의 행위가 같기 때문이다. 한편 $M \lesssim M'$인 경우, ACTL[2] 논리식 ϕ에 대해서 $M' \models \phi \Rightarrow M \models \phi$이 성립한다. 즉 M'에서 ϕ가 만족되면 M에서도 ϕ가 만족한다. 하지만 반대의 경우는 성립하지 않는다. 왜냐하면 M'의 행위가 M보다 더 크기 때문이다. 마찬가지로 $M \lesssim M'$인 경우, ECTL 논리식 ϕ에 대해서 $M \models \phi \Rightarrow M' \models \phi$이 성립한다. 즉 M에서 ϕ가 만족되면 M'에서도 ϕ가 만족된다. 하지만 반대의 경우는 성립하지 않는다. 왜냐하면 M의 행위가 M'보다 더 적기 때문이다.

3. 상태 투영을 통한 추상화

이제 본 논문에서 제안하는 상태 축소 방법을 제시한다. 모델의 상태 공간은 모델을 정의하는 상태 변수들에 의해 구성된다. 모델을 정의하는 상태 변수 집합을 $V=\{v_1, \cdots, v_n\}$라고 하자. 각 변수 v_i가 가질 수 있는 값의 범위를 도메인 D_i라 부르며, 변수는 도메인 값 중의 하나인 $d_i \in D_i$를 항상 갖는다. 즉,

$$v_i = d_i$$

이다. $\sigma_i : \{v_i\} \rightarrow D_i$는 변수 v_i에 값을 배정하는 함수로서 아래와 같이

$$\sigma_i(v_i) = d_i \in D_i$$

정의된다. $\sigma : \{v_1\} \times \cdots \times \{v_n\} \rightarrow D_1 \times \cdots \times D_n$는 상태를 나타내는 함수로서 아래와 같이

$$\sigma(v_1, \cdots, v_n) = (\sigma_1(v_1), \cdots, \sigma_n(v_n)) = (d_1, \cdots, d_n)$$

정의된다. $(d_1, \cdots, d_n)$를 모델의 임의의 상태 s라고 하자. 모델의 전체 상태 공간은 상태 s의 모임이며, 따라서 모델이 변수 집합 V를 가질때 모델의 전체 상태 공간은 $S = D_1 \times \cdots \times D_n$이다. 변수의 부분 집합 $V' \subseteq V$(여기서 부분 집합의 크기는 $m = |V'|$이며 $m < n$)이 주어졌을때, 상태 s를 집합 V'으로 투영하는 하는 것은 다음과 같이

$$proj(s, V') = (d_1, \cdots, d_m) \text{ such that } \sigma_i^{-1}(d_i) \in V' \text{ for all } i$$

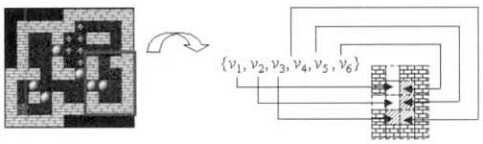
정의된다. 상태들의 집합 S를 V'으로 투영하는 것은 다음과 같이 정의된다.

$$proj(S, V') = \{proj(s, V') | s \in S\}$$

V로 정의된 상태 공간을 V'으로 투영시키면 전체 공간은 $D_1 \times \cdots \times D_n$에서 $D_1 \times \cdots \times D_m$으로 축소된다. 즉 $2^{|V|} - 2^{|V'|}$ 만큼의 공간이 줄어든다. 따라서 주어진 논리식을 모델 체킹할 때 필요한 변수(V')와 불필요한 변수($V - V'$)로 변수 집합 V를 먼저 구분한 후에, 전체 상태 공간을 필요한 변수들로 투영 시켜야 한다. 그러므로 주어진 논리식의 모델 체킹과 관련된 변수를 찾는 것이 상태 축소, 즉 추상화의 기본이다. 모델 체킹시 필요한 변수를 파악하기 위해서는 도메인 지식이 필요하다. 본 논문에서 다루는 모델 M은 푸쉬 푸쉬 게임

2) CTL 논리식의 일부분으로서 "모든 경로"를 나타내는 A가 시제 연산자 앞에 나오는 논리식이다.

을 나타낸다. 주어진 공을 목표 지점으로 모두 이동시키는 최단 경로를 찾는 것이 게임의 목표이며, 본 논문에서는 모델 체킹을 이용해서 그러한 경로를 구한다. (그림 1)에서 보듯이 푸쉬 푸쉬 게임에는 많은 셀들이 있다. 셀에는 공이 놓일수도 있고 비어있을수도 있기 때문에, 셀은 이진 변수로 모델링된다. 셀이 n개 있는 경우 게임의 상태 공간은 이진 변수 집합 $V=\{v_1,\cdots,v_n\}$로 정의되고 각 변수에 값을 배정하는 함수는 $\sigma_i : \{v_i\} \rightarrow B$ 로 정의된다. (그림 1)에서 네모 상자로 표시한 우측 하단을 살펴보자. 6개의 셀을 표현하기 위해서 6개의 이진 변수가 배정되었다. v_1, v_2에 해당하는 셀에는 공이 들어갈 수 있거나 있던 공이 나올수 있다. 그러나 v_3, v_4, v_5, v_6가 가르키는 셀에는 공이 한 번 들어가면 다시 나올수 없다. 이와같은 셀은 최단 경로와는 무관하다.



(그림 1) 푸쉬 푸쉬 게임

게임을 모델링하는 상태 변수 집합 V를 두개의 집합으로 분할할 수 있다.

$$V = V_p \cup V_d$$

여기서 V_p는 공의 진출입이 자유스러운 셀을 나타내고, V_d는 공이 한 번 들어오면 다시 나갈 수 없는 셀을 나타낸다. 모델의 상태 공간을 최단 경로와 관련된 집합 V_p로 투영함으로서 상태 공간을 축소할 수 있다. V_d를 데드락 변수의 모임이라고 하자. 함수 $sel(S, V_d)$는

$$sel(S, V_d)=\{(d_1,\cdots,d_n) \mid \text{there is some } d_i \text{ such that } \sigma_i^{-1}(d_i) \in V_d \wedge d_i = 1\}$$

데드락 변수에 공이 위치하는 상태를 리턴하는 함수이다. 최단 경로와 무관한 데드락 변수를 제거함으로서 모델의 상태 공간을 축소할 수 있다. 다시말해서, 최단 경로와 관련된 집합 V_p로 투영함으로서 상태 공간을 축소할 수 있다. $V'=V_p$라고 하자. 원래 모델 M을 V'으로 투영하면 다음과 같은 추상화 모델 $M'=<S', I', R', AP', L'>$를 얻는다.

- $S' = proj(S-sel(S, V_d), V')$
- $I' = proj(I, V')$
- $R' = \{(s, s') \mid s=proj(s_1, V'),\ s'=proj(s_2, V'),\ (s_1, s_2) \in (R \setminus (sel(S, V_d)\times S) \setminus (S\times sel(S, V_d)))\}$
- $L'(s') = \{q \mid s' = proj(s, V') \wedge s \models q\} \cap AP'$

전이 관계의 정의에서 사용된 $R \setminus (sel(S, V_d)\times S) \setminus (S\times sel(S, V_d))$ 식은 원래 전이 관계에서 데드락 변수와 관계된 모든 전이를 제거하는 식이다. 투영에 의해서 상태 공간의 수가 축소되었을뿐만 아니라 원래보다 적은 행위를 갖는다. 따라서 추상화 모델 M'과 원래 모델 M은 시뮬레이션 관계이다.

[정리 1] $M' \lesssim M$

[증명] 증명을 돕기 위해 중간 모델 $M^{\downarrow}=<S, I, R^{\downarrow}, AP, L>$을 이용한다. $M^{\downarrow}$은 원래 모델 M으로부터 구한다. 여기서 $R^{\downarrow} = R \setminus (sel(S, V_d)\times S) \setminus (S\times sel(S, V_d))$이다. $R^{\downarrow}$만 다를 뿐 나머지는 M과 같다. $R^{\downarrow} \subseteq R$이기 때문에 $M^{\downarrow}$의 행위가 M보다 적다. 따라서 $M^{\downarrow} \lesssim M$이다. 위에서 정의한 추상화 모델 M'과 중간 모델 $M^{\downarrow}$은 상태 표현이 축소된 것을 제외하고는 같다. $M^{\downarrow}$의 행위가 M'에서 그대로 보존되고, 반대의 경우도 마찬가지이다. 따라서 $M^{\downarrow} \equiv M'$이다. $M^{\downarrow} \lesssim M$이고 $M' \equiv M'$이기 때문에 $M' \lesssim M$이다.

전장에서 설명한 바와 같이 $M' \lesssim M$인 경우, ECTL 논리식 ϕ에 대해서 $M' \models \phi \Rightarrow M \models \phi$이 성립한다. 즉 M'에서 ϕ가 만족되면 M에서도 ϕ가 만족된다. 왜냐하면 M'의 행위가 M보다 더 적기 때문이다. 게임의 목표 상태를 $goal$ 이라고 하자. "목표 상태에 도달가능하다"를 ECTL 논리식으로 표현하면 EF$goal$이다. 따라서 원래 모델 대신 상태 공간이 축소된 추상화 모델을 사용해서 모델 체킹할 수 있다.

[정리 2] $M' \models \text{EF}goal \Rightarrow M \models \text{EF}goal$

[증명] ① $M' \lesssim M$ 정리 1

② $M \models \text{AG}\Box goal \Rightarrow M' \models \text{AG}\Box goal$

1번과 ACTL식의 만족 관계([9]참조)

③ $M' \not\models \text{AG}\Box goal \Rightarrow M \not\models \text{AG}\Box goal$

2번의 대우

④ $M' \models \Box\text{AG}\Box goal \Rightarrow M \models \Box\text{AG}\Box goal$

3번의 불만족 관계를 만족 관계로 변환

⑤ $M' \models \text{EF}goal \Rightarrow M \models \text{EF}goal$

4번의 부정 정규형(negation normal form)

4. 실 험

본 논문에서 제시한 상태 축소 방법은 다양한 분야에 적용가능하다. 추상화를 통해서 얼만큼의 성능 개선이 있었는지를 실제 확인하기 위해서 푸쉬 푸쉬 게임 풀이에 추상화 기법을 적용하였다. 푸쉬 푸쉬 게임은 일종의 도달성 문제로 간주할 수 있기 때문에, 모델 체킹으로 게임을 풀 수 있는 최단 경로를 얻을 수 있다. 주어진 게임으로 부터, 모델 체커에 입력될 유한 상태 모델과 CTL 논리식을 자동 추출할 수 있다. 만약 초기 상태에서 목표 상태로 도달되는 경로가 존재한다면, 모델 체커는 게임을 풀 수 있는 최단 경로를 출력한다(지면 관계상 게임 풀이와 관련된 유한 상태 모델, CTL 논리식, 최단 경로 찾기등의 설명은 생략한다. 이들에 대한 상세한 내용은 관련 연구[6, 10]을 참조하기 바람).

실험을 위해서 두 개의 모델 체커 Cadence SMV, NuSMV를 사용하여 푸쉬 푸쉬 게임을 풀었다. 게임 풀이에 소요된 시간과 메모리 사용량이 <표 1>에 있다. 표에서 기호 ∞은 3시간 이상 경과해도 결과를 볼 수 없었음을 나타낸다. 다시 말해서, 상태 폭발이 일어난 경우를 의미한다. 실험은 펜티엄 4에서 1기가 메모리를 가진 컴퓨터에서 수행하였다. 먼저 Cadence SMV[7]를 사용한 경우를 살펴보자. 처음에는

추상화없이 그냥 모델 체킹을 사용하여 게임을 풀었다. 그 결과 전체 50게임 중에서 37게임은 풀었으나 나머지 13게임은 상태 폭발 때문에 풀지 못했다. 그래서 추상화를 적용하여 모델을 축소한 후에 다시 모델 체킹을 수행하였다. 그 결과, 추상화 이전에 비해서 13%의 시간과 21%의 메모만을 사용하고서도 37게임에 대해서 똑 같은 결과를 얻었다. 즉 87%의 시간 절감 및 79%의 메모리 절감 효과가 있었다. 뿐만 아니라, 상태 폭발 문제로 인해서 풀이 경로를 찾지 못했던 13게임중에서 12게임을 성공적으로 해결하였다. NuSMV [8]의 결과도 비슷했다. 추상화없이 전체 50게임 중에서 43게임을 풀었으나, 나머지 7게임은 상태 폭발 때문에 풀지 못했다. 추상화를 적용한 경우 추상화 이전에 비해서 17%의 시간과 44%의 메모만을 사용하고서도 43게임에 대해서 똑 같은 결과를 얻었다. 즉 83%의 시간 절감 및 56%의 메모리 절감 효과가 있었다. 뿐만 아니라, 상태 폭발 문제 때문에 풀지 못했던 7게임 중에서 6게임을 추상화로 해결하였다. 실험 결과에서 보듯이 추상화는 시간과 메모리 절감을 제공할 뿐만아니라 크기가 큰 모델의 모델 체킹을 가능케 했다.

〈표 1〉 푸쉬푸쉬 게임 수행 결과

판	Cadence SMV 수행 결과				NuSMV 수행 결과			
	추상화 이전		추상화 이후		추상화 이전		추상화 이후	
	시 간 (초)	메모리 (MB)	시 간 (초)	메모리 (MB)	시 간 (초)	메모리 (MB)	시 간 (초)	메모리 (MB)
1	0.6	2	0.1	1	0.1	2	0.1	2
2	3.6	6	0.5	5	2.4	7	0.6	4
3	33.8	32	2.6	7	18.0	13	4.9	12
4	0.7	3	0.5	1	1.7	8	0.6	5
5	1.0	3	0.3	1	1.4	5	0.3	3
6	123.3	93	7.5	11	73.6	16	12.8	12
7	53.6	37	8.1	13	29.5	13	4.5	12
8	∞		53.9	105	6,365.1	372	260.2	26
9	11.0	11	0.3	1	5.9	12	0.3	3
10	5.0	8	0.4	1	7.1	13	1.5	6
11	6.2	8	0.3	1	5.1	11	0.6	4
12	10.7	12	1.5	6	8.7	12	1.7	7
13	355.3	240	0.4	1	83.8	25	0.5	4
14	125.8	80	2.3	7	143.7	24	3.1	12
15	∞		167.8	190	3,250.7	210	321.6	19
16	16.5	16	0.5	1	14.4	13	0.6	4
17	∞		415.0	470	3,487.7	211	117.6	14
18	1200.0	508	8.1	21	711.8	59	1.8	8
19	1200.0	613	209.0	325	1,166.5	108	479.4	33
20	∞		224.0	358	7,516.6	342	404.2	22
21	2880.0	1109	14.8	26	1,610.0	89	50.5	14
22	420.0	219	52.5	59	354.0	37	103.1	14
23	3000.0	870	79.0	67	1,777.0	103	50.5	14
24	600.0	317	7.1	17	581.6	56	13.8	12
25	10.0	305	13.5	25	410.3	43	78.8	16
26	60.0	34	5.4	8	33.3	13	10.1	12
27	30.0	20	3.0	7	18.9	13	4.7	12
28	420.0	210	4.8	13	356.2	40	11.9	13
29	16.0	13	1.4	5	8.6	12	2.5	9
30	∞		248.1	1	∞		2,990.7	171
31	1020.0	500	8.3	16	662.3	63	27.8	14
32	∞		9.9	20	1,523.1	73	31.2	13
33	∞		168.2	298	∞		1,260.5	117
34	∞		101.8	113	∞		3,039.4	448
35	∞		98.6	133	∞		1,210.9	119
36	∞		73.6	122	∞		211.2	19
37	2200.0	854	7.7	18	500.5	63	19.4	13
38	28.0	18	3.0	8	231.5	19	8.7	13
39	960.0	336	10.4	11	162.1	34	13.0	13
40	∞		4.3	13	6,594.8	685	442.4	78
41	120.0	58	3.4	10	77.7	14	2.5	10
42	1260.0	532	25.0	40	604.4	68	68.1	13
43	480.0	299	223.3	272	1,639.1	130	1,627.0	103
44	1860.0	539	12.9	18	1,116.2	116	53.9	16
45	300.0	109	6.5	13	92.3	19	6.4	12
46	780.0	413	10.2	17	180.3	32	30.1	13
47	35.0	25	1.8	5	20.1	13	3.7	11
48	1980.0	841	16.6	35	492.6	47	42.2	14
49	∞		234.2	321	∞		2,663.0	228
50	∞		∞		∞		∞	

5. 관련 연구들

상태 폭발 문제는 모델 체킹의 가장 큰 장애물이기 때문에 이를 해결하기 위해 추상화에 관한 연구가 매우 많이 있었다. 이들 연구들은 크게 바이시뮬레이션에 관한 연구와 시뮬레이션에 관한 연구로 구분할 수 있다. 바이시뮬레이션에 관한 대표적인 연구는 COI(Cone Of Influence) 축소이다[9]. COI 축소에서는 검사할 논리식에 나타난 변수와 이들 변수에게 영향을 주는 변수들을 종속성 분석으로 구한다. 그런 후에, 주어진 논리식을 모델 체킹하는데 필요한 변수만을 남기고 불필요한 변수를 삭제하면 원래 모델 M보다 축소된 모델 M'을 얻게 된다. COI 축소는 검사할 논리식의 관점에서 추상화를 하기 때문에 CTL 논리식 ϕ의 관점에서 두 모델의 행위는 일치한다. 즉 $M \equiv M'$로서 바이시뮬레이션 관계에 있다. 본 논문에서는 바이시뮬레이션이 아니라 시뮬레이션 관계에 있는 추상 모델을 구한다.

시뮬레이션에 관한 연구들은 매우 많다. 대부분의 추상화 연구들은 이 범주에 속한다. 대표적인 연구로는 존재(existential) 추상화가 있다[11]. 모델에 존재하는 자료들간의 동치 관계를 이용하여 자료를 동치 클래스로 분할한 후, 동치 클래스를 하나의 값으로 대표하게 함으로서 모델 체킹에서 사용되는 자료의 범위를 크게 축소할 수 있다. 따라서 원래 모델 M과 추상화된 모델 M'은 $M \preceq M'$ 관계에 있다. 본 논문에서의 목표는 자료 추상화가 아니라 상태 전이 시스템의 추상화이다. 시뮬레이션에 관한 또 다른 연구로는 반례를 이용한 자동 추상화가 있다[12]. 여기서는 원래 모델에 대해 시뮬레이션 관계에 있는 추상화된 모델 M'을 구한다. ACTL 논리식 ϕ에 대해서 $M' \models \phi$이면 $M \models \phi$이다. 그러나, $M' \not\models \phi$인 경우 원래 모델에서의 결과를 알 수 없다. 그래서 출력된 반례를 살펴보아야 한다. 만일 반례가 원래 모델에 존재한다면 $M \not\models \phi$라고 판정할 수 있다. 그러나 반례가 원래 모델에 존재하지 않는 경우, 결과를 단정할 수 없다. 그래서 반례를 이용해서 새로운 모델 M''을 생성해서(이때 M, M', M''의 관계는 $M \preceq M'' \preceq M'$) 위의 과정을 반복한다. 반례를 이용해서 추상화 모델을 구하는 이들 연구와는 달리, 본 연구에서는 변수의 위치 정보를 이용해서 추상화 모델을 구하며 반례는 게임의 최단 풀이 경로를 찾는데 사용된다[13].

시뮬레이션에 관한 또 다른 연구로는 투영을 이용한 추상화 연구가 있다[14]. 모델 체킹에 필요한 변수들로 상태 공간을 투영함으로서 원래 모델 M에 대한 추상화 모델 M'을 구한다. 상태가 축소 통합되는 과정에서 행위가 증가되어서 원래 모델에 비해 상위 근사화(over approximation) 관계에 있는 추상 모델 $M \lesssim M'$이 생성된다. 이와는 달리 본 논문에서는 $R \setminus (sel(S,V_d) \times S) \setminus (S \times sel(S,V_d))$을 이용하여 불필요한 행위를 제거한 후 투영을 하기 때문에 원래 모델에 비해서 행위가 줄어든 하위 근사화(under approximation) 관계에 있는 모델 $M' \lesssim M$이 생성된다.

6. 결 론

모델 체킹이 산업체에 활용되기 위해서는 크기 문제를 극복해야 한다. 산업체에서 사용되는 모델의 크기는 매우 큰 반면에, 현재의 모델 체킹 기술로 처리할 수 있는 모델의 크기는 적다(이진 변수 약 100~150개로 구성된 모델을 현재 처리할 수 있다). 따라서 모델 체킹의 주요한 연구 분야중의 하나가 크기 문제의 극복, 즉 상태 폭발 문제 해결이다.

본 논문에서는 상태 폭발 문제를 방지하기 위해서, 상태 공간의 투영을 통해서 추상화 모델 M'을 얻는 방법을 제안했다. 제안한 방법으로 얻어진 추상화 모델 M'은 원래 모델 M보다 크기가 작을뿐만 아니라 더 적은 행위를 갖는다. 따라서 $M' \lesssim M$이다. 적은 행위를 갖는 M'에서 ECTL 논리식이 만족되면 더 큰 행위를 갖는 M에서도 ECTL 논리식이 만족된다는 사실을 이용해서, 푸쉬 푸쉬 게임을 풀 수 있었다. 왜냐하면 푸쉬 푸쉬 게임은 일종의 도달성 문제이며, 도달성 문제는 ECTL 모델 체킹으로 해결할 수 있기 때문이다. Cadence SMV와 NuSMV를 이용하여 실험한 결과, 상태 폭발 문제로 풀 수 없었던 대부분의 게임을 추상화를 이용해서 해결하였다. 그리고 추상화를 적용하기 이전에 비해서 시간 절감 및 메모리 절감 효과가 있었다. Cadence SMV의 경우 평균 87%의 시간 절감 및 79%의 메모리 절감이 있었으며, NuSMV의 경우 83%의 시간 절감 및 56%의 메모리 절감이 있었다.

그러나 <표 1>의 마지막 행에서 보듯이 추상화를 적용해서도 50번째 게임을 풀 수 없었다. 50번째 게임은 푸쉬 푸쉬 게임에서 가장 복잡한 게임이다. 비록 "분활 후 정복"의 원리를 이용한 릴레이 모델 체킹으로 50번째 게임을 풀었지만 그 결과가 최단 경로라고 보장할 수 없다[15]. 왜냐하면 한 번의 모델 체킹으로 얻어낸 결과가 아니라, 여러 번 모델 체킹한 결과를 결합한 것이기 때문이다. 따라서 한 번의 모델 체킹으로 최단 경로를 구할 수 있도록 상태 공간을 크게 축소하는 추상화 기법을 연구해야 한다.

여기서는 게임을 예로 삼아 추상화의 효과를 설명하였다. 앞으로 소프트웨어 개발과 관련된 모델(예를 들면 UML의 상태 천이 다이아그램)에 제안한 추상화 기법을 적용하여 소프트웨어 모델을 효과적으로 정형 검증하는 연구가 필요하다.

참 고 문 헌

[1] K. L. McMillan, "Symbolic Model Checking : An Approach to the State Explosion Problem," PhD thesis, Carnegie Mellon University, Department of Computer Science, 1992.

[2] R. Bloem, I. Moon, K. Ravi and F. Somenzi, "Approximations for Fixpoint Computations in Symbolic Model Checking," in Proceedings of *SCI'2000,* Vol.VIII, Part II, 2000, pp.701-706, 2000.

[3] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith, "Progress on the State Explosion Problem in Model Checking," in Proceedings of 10 Years Dagstuhl, LNCS 2000, pp.154-169, 2000.

[4] Y. Lu, "Automatic Abstraction in Model Checking," Ph.D. thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, 2000.

[5] R. Bloem, K. Ravi and F. Somenzi, "Symbolic Guided Search for CTL Model Checking," in Proceedings of Design Automation Conference, pp.29-34, 2000.

[6] G. Kwon, "Applying Model Checking Techniques to Push Push Game Solving," in Proceedings of SERA2003, LNCS 3026, pp.290-303, 2003.

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

[8] http://nusmv.irst.itc.it/.

[9] E. M. Clarke, O. Grumberg and D. Peled, Model Checking, MIT Press, 1999.

[10] 권기현, "모델 검증을 이용한 게임 풀이", 정보과학회학회지, 제21권 제1호, pp.7-14, 2003.

[11] E. M. Clarke, O. Grumberg and D.E. Long, "Model Checking and Abstraction," ACM Transactions on Programming Languages and Systems, Vol.16, No.5, pp.1512-1542, 1994.

[12] E. M. Clarke, O. Grumberg, S. Jha, Y. Lu and H. Veith, "Counterexample-Guided Abstraction Refinement," in Proceedings of Computer Aided Verification, pp.154-169, 2000.

[13] 권기현, 이태훈, "게임 풀이를 위한 NuSMV의 효율적인 반례 생성," 한국정보처리학회, 정보처리학회논문지D, 제10-D권 제5호, pp.813-820, 2003.

[14] S. Barner, D. Geist and A. Gringauze, "Symbolic Localization Reduction with Reconstruction Layering and Backtracking," in Proceedings of CAV'02, LNCS 2404, pp.65-77, 2002.

[15] T. Lee, G. Kwon, "Relay Model checking for Avoiding The State Explosion Problem," In Proceedings of SERA'2004, pp.305-310, 2004.

권 기 현

e-mail : khkwon@kyonggi.ac.kr

1985년 경기대학교 전자계산학과(학사)

1987년 중앙대학교 전자계산학과 (이학석사)

1991년 중앙대학교 전자계산학과 (공학박사)

1998년~1999년 독일 드레스덴 대학 전자계산학과 방문교수

1999년~2000년 미국 카네기 멜론 대학 전자계산학과 방문교수

1991년~현재 경기대학교 정보과학부 교수

관심분야 : 소프트웨어 모델링, 소프트웨어 분석, 정형 기법 등